A Evaluation of (E1) Correctness and Completeness

In the following, we focus on proving that the presented approach for the adaptation of process graphs fulfills the properties correctness, completeness and termination. In our approach, we state to "conduct planning steps". Here, we make use of existing techniques for the automated planning of process models [e.g., Bertoli et al. 2006; Heinrich et al. 2015] and refer to their works for providing proofs for the fact that planning by conducting planning steps fulfills the properties correctness, completeness and termination.

THEOREM 1. *The process graphs constructed by the approach are correct: Only feasible paths are contained in an adapted process graph.*

PROOF. We distinguish the possible changes as described in the approach.

Updating the initial state. We show the feasibility of each path by using Definition 5 and, in particular, conditions i. to iii. To satisfy condition iii. of Definition 5, we examine all state transitions within the paths of the adapted process graph. In accordance with our approach, these transitions can be divided into transitions from old, updated and new states.

For each new and updated state, the applicability of each following action in the adapted process model is verified and the following states are constructed by (partially) applying the transition function where needed, which leads to feasibility condition iii. being fulfilled by construction. Additionally, old states stem from the given process graph. Since only the initial state was updated, no further changes occur once an old state has been reached and thus the according subgraphs, which remain correct, are used for the adapted process graph. Hence, the feasibility condition iii. is fulfilled in these cases as well.

A path is completed as soon as we are certain to reach a goal state through an old, updated or new state. When no goal state is reached and no further actions can be applied, the current path is not considered in the adapted process graph. Thus, conditions i. and ii. of Definition 5 are fulfilled and the feasibility of every path in the adapted process graph is proven.

Adding a goal state. At first, we consider all paths feasible in the given process graph. For such a path $(bs_{init},a_1,bs_2,...,bs_k)$, two cases may occur: In the first case, $bs_{init},bs_2,..., bs_{k-1}$ do not meet the new goal state. Then, the path remains feasible as the conditions for a feasible path in Definition 5 are still fulfilled. In the second case, (at least) one of the belief states $bs_{init},bs_2,...,bs_{k-1}$ meets the new goal state *goal*. In this case, the existing path is shortened until only one such belief state is contained and is the last belief state in the path. This shortened path is feasible as well: i. and ii. in Definition 5 are fulfilled by construction and iii. remains fulfilled. In addition to these kinds of paths, the adapted process graph may also contain new paths leading to *goal*. As we construct these paths by conducting planning steps (i.e., iteratively computing *app(bs)* and applying the transition function for every belief state *bs*), the paths are always feasible.

Removing a goal state. We again consider all paths feasible in the given process graph. Let $(bs_{init},a_1,bs_2,...,bs_k)$ be such a path. If bs_k meets an element of $GOALS'=GOALS\setminus\{goal\}$, the path obviously remains feasible as the conditions in Definition 5 are still fulfilled. On the other hand, if the path had ended at *goal* it is extended until a belief state meeting a goal state from *GOALS'* is reached. This is done by conducting planning steps until the extended path is feasible. If such an extension to a feasible path is not possible, the path, which formerly was feasible, is not considered in the adapted process graph. No further new paths are constructed. Hence, overall, only feasible paths are contained in the adapted process graph.

Updating a goal state. Strengthening update. Again, we consider all paths feasible in the given process graph. Let $(bs_{init},a_1,bs_2,...,bs_k)$ be such a path. If bs_k meets a goal state from $GOALS\setminus\{goal\}$, the path obviously remains feasible as the conditions in Definition 5 are still fulfilled. If the path had ended at *goal*, it is extended until a belief state meeting a goal state from *GOALS'* is reached. This is done by conducting planning steps until the extended path is feasible. If such an extension to a feasible path is not possible, the path, which formerly was feasible, is not considered in the adapted process graph. No further new paths are constructed. Thus, similar to the case of removing a goal, only feasible paths are contained in the adapted process graph.

Updating a goal state. Weakening update. Yet again, we first consider all paths feasible in the given process graph. For such a path $(bs_{init},a_1,bs_2,...,bs_k)$, two cases may occur: In the first case, $bs_{init},bs_2,...,bs_{k-1}$ do not meet *goal'*. Then, the path remains feasible as the conditions for a feasible path in Definition 5 are still fulfilled. In the second case, (at least) one of the belief states $bs_{init},bs_2,..., bs_{k-1}$ meets *goal'*. In this case, the existing path is shortened until only one such belief state is contained and is the last belief state in the path. This shortened path is feasible as well: i. and ii. in Definition 5 are fulfilled by construction and iii. remains fulfilled. In addition to these kinds of paths, the adapted process graph may also contain new paths leading to *goal'*. As we construct these paths by conducting planning steps, the paths are always feasible.

Adding an action. At first we note that all paths of the given process graph remain feasible when adding *a* to the set of actions since the conditions i. to iii. of Definition 5 are still fulfilled. In addition to that we try to reach goal states through new paths that include the action *a*. As we do this by conducting planning steps, the constructed paths are always ensured to be feasible.

Removing an action. When removing an action *a* from the set of actions, the adapted process graph contains the feasible paths of the given process graph which do not contain *a*. As every action in such a path still fulfills the applicability criterion and the last belief state is the only belief state that meets a goal state in such a path, these paths remain feasible (cf. Definition 5).

Updating an action. Strengthening update of the preconditions. The adapted process graph consists of paths retained from the given process graph and newly constructed paths. We start by considering all paths feasible in the given process graph. If such a path $(bs_{init},a_1,bs_2,...,bs_k)$ does not contain *a* at all, it obviously remains feasible as the path does not change at all and the conditions i.-iii. in Definition 5 are still fulfilled. Otherwise, there exists $i<k$ with $a = a_i$ and the applicability of *a'* in bs_i is checked. If $a' \in app(bs_i)$ and $R(bs_i,a) \neq R(bs_i,a')$, one tries to construct a new path and proceeds as when treating the case of the initial state, for which correctness was already proven. Hence, all newly constructed paths are feasible. On the other hand, if $a' \in app(bs_i)$ and $R(bs_i,a) = R(bs_i,a')$ for all *i*, the conditions i.-iii. in Definition 5 remain fulfilled as all belief states in the path remain identical. Finally, if $a' \notin app(bs_i)$, the path is not considered in the adapted process graph. Hence, only feasible paths are retained from the given process graph.

Updating an action. Weakening update of the preconditions. In case of a weakening update of the preconditions of an action *a* resulting in the action *a'*, all feasible paths of the given process graph which do not contain *a* are retained. Since these paths remain unchanged, they stay feasible. Furthermore, additional paths in the adapted process graph are retrieved by conducting planning steps from belief states *bs* with $a \notin app(bs)$; they are feasible by construction. Additionally, for every belief state *bs* of the given process graph with $a \in app(bs)$ and $a' \in app(bs)$, we follow the approach for an update of the initial state. This approach, as seen above, leads to feasible paths.

Updating the effects. When updating the effects of an action *a*, one retains all paths $(bs_{init},a_1,bs_2,...,bs_k)$ which do not contain *a* at all and obviously remain feasible as they do not change at all. Otherwise, for $a = a_i$ $(i<k)$, if $R(bs_i,a) \neq R(bs_i,a')$, one tries to construct a new path and proceeds as when treating the case of the initial state, for which correctness was already proven.

Thus, Theorem 1 is shown for each case and hence proven. *q.e.d.*

THEOREM 2. *The process graphs constructed by the approach are complete: All feasible paths are contained in an adapted process graph.*

PROOF. We distinguish the possible changes as described in the approach.

Updating the initial state. According to Definition 5, each feasible path has to start with the initial state and each following action has to be applicable in its preceding state and has to lead to a goal state (in the sense that the conduction of this action and potentially subsequent actions results in a goal state). As in the previous theorem, we therefore examine all state transitions from old, updated and new states in order to show that the adapted process graph contains every path of this nature. Let *bs* be a belief state.

In case of *bs* being an old state, we retain the subgraph from the given process model which starts with *bs*. As the given process graph is complete, this subgraph contains all actions from *app(bs)* that lead to a goal state.

Let *bs* be an updated or new state. If *bs* itself does not meet a goal state, each action $a \in app(bs)$ is checked in regard to whether it leads to a goal state: For each such action *a* all possible subsequent actions and states are retrieved by applying the transition function until a goal state is reached, an old state is reached or no further action can be applied. In the first two cases, *a* indeed leads to (at least) one goal state and consequently the sequence *bs,a* is part of a feasible path. In the last case, on the contrary, every path containing the sequence *bs,a* is not considered in the adapted process graph as *a* does not lead to a goal state.

To sum up, beginning with bs_{init} for each old, updated or new state we either examine all applicable actions and thereafter retain those leading to a goal state or we retain a subgraph of the given process graph which contains all applicable actions that lead to a goal state. Thus, the adapted process graph contains all feasible paths.

Adding a goal state. Let $(bs_{init},a_1,bs_2,...,bs_k)$ be any feasible path after the addition of *goal* to the set of goal belief states *GOALS* such that $GOALS'=GOALS\cup\{goal\}$. We need to show that $(bs_{init},a_1,bs_2,...,bs_k)$ is indeed contained in the adapted process graph. As $(bs_{init},a_1,bs_2,...,bs_k)$ is feasible, according to Definition 5 i., bs_k meets one of the goal states from $GOALS'=GOALS\cup\{goal\}$. In the first case, let bs_k meet one of the goal states from *GOALS*. In this case, $(bs_{init},a_1,bs_2,...,bs_k)$ was feasible in the given process graph and - since bs_{init},bs_2,bs_{k-1} do not meet a goal state from *GOALS'* because of Definition 5 ii. - is retained from the given process graph and thus contained in the adapted process graph. In the second case, bs_k meets *goal*. It is then possible that $(bs_{init},a_1,bs_2,...,bs_k)$ was part of a feasible path $(bs_{init},a_1,bs_2,...,bs_k,...,bs_m)$ with $m>k$ in the given process graph. Such a path $(bs_{init},a_1,bs_2,...,bs_k,...,bs_m)$ is, according to the approach, shortened to $(bs_{init},a_1,bs_2,...,bs_k)$ and then retained. Hence, $(bs_{init},a_1,bs_2,...,bs_k)$ is contained in the adapted process graph. If $(bs_{init},a_1,bs_2,...,bs_k)$ was not part of a feasible path $(bs_{init},a_1,bs_2,...,bs_k,...,bs_m)$ in the given process graph, (at least) one of the actions $a_1,a_2,...$ was not planned in the given process graph in bs_{init}, resp. $bs_1,...$. For such actions, planning steps are conducted, which lead to the inclusion of $(bs_{init},a_1,bs_2,...,bs_k)$ into the adapted process graph. Thus, overall, it is guaranteed that $(bs_{init},a_1,bs_2,...,bs_k)$ is contained in the adapted process graph.

Removing a goal state. Let $(bs_{init},a_1,bs_2,...,bs_k)$ be any feasible path after the removal of *goal* from the set of goal belief states *GOALS* such that $GOALS'=GOALS\setminus\{goal\}$. If $bs_{init},bs_2,...,bs_{k-1}$ do not meet *goal*, $(bs_{init},a_1,bs_2,...,bs_k)$ was a feasible path in the given process graph (cf. conditions i.-iii. in Definition 5), which is retained and hence contained in the adapted process graph. Otherwise, let bs_m ($m<k$) be the first belief state that meets *goal*. Then, $(bs_{init},a_1,bs_2,...,bs_m)$ was a feasible path in the given process graph which, by according to our approach, is extended to $(bs_{init},a_1,bs_2,...,bs_k)$ – and possibly other additional feasible paths – in the adapted process graph.

Updating a goal state. Strengthening update. Let $(bs_{init},a_1,bs_2,...,bs_k)$ be any feasible path after the update of a goal state *goal* to *goal'*. If $bs_{init},bs_2,...,bs_{k-1}$ do not meet *goal*, $(bs_{init},a_1,bs_2,...,bs_k)$ was a feasible path in the given process graph (cf. conditions i.-iii. in Definition 5), which is retained and hence contained in the adapted process graph. Otherwise, let bs_m be the first such belief state. Then, $(bs_{init},a_1,bs_2,...,bs_m)$ was a feasible path in the given process graph which, according to our approach, is extended to $(bs_{init},a_1,bs_2,...,bs_k)$ – and possibly other additional feasible paths – in the adapted process graph.

Updating a goal state. Weakening update. Let $(bs_{init},a_1,bs_2,...,bs_k)$ be any feasible path after the update of a goal state *goal* to *goal'*. We need to show that $(bs_{init},a_1,bs_2,...,bs_k)$ is indeed contained in the adapted process graph. As $(bs_{init},a_1,bs_2,...,bs_k)$ is feasible, according to Definition 5 i., bs_k meets one of the goal states from $GOALS'=(GOALS\setminus\{goal\})\cup\{goal'\}$). In the first case, let bs_k meet one of the goal states from $GOALS\setminus\{goal\}$. In this case, $(bs_{init},a_1,bs_2,...,bs_k)$ was feasible in the given process graph and - since bs_{init},bs_2,bs_{k-1} do not meet a goal state from *GOALS'* because of Definition 5 ii. - is retained from the given process graph and thus contained in the adapted process graph. In the second case, bs_k meets *goal'*. It is then possible that $(bs_{init},a_1,bs_2,...,bs_k)$ was part of a feasible path $(bs_{init},a_1,bs_2,...,bs_k,...,bs_m)$ with $m\geq k$ in the given process graph. Such a path $(bs_{init},a_1,bs_2,...,bs_k,...,bs_m)$ is shortened to $(bs_{init},a_1,bs_2,...,bs_k)$ and then retained. Hence, $(bs_{init},a_1,bs_2,...,bs_k)$ is contained in the adapted process graph. If $(bs_{init},a_1,bs_2,...,bs_k)$ was not part of a feasible path $(bs_{init},a_1,bs_2,...,bs_k,...,bs_m)$ in the given process graph, (at least) one of the actions $a_1,a_2,...$ was not planned in the given process graph in bs_{init}, resp. $bs_2,...$. For such actions, planning steps are conducted, which lead to the inclusion of $(bs_{init},a_1,bs_2,...,bs_k)$ into the adapted process graph. Thus, overall, it is guaranteed that $(bs_{init},a_1,bs_2,...,bs_k)$ is contained in the adapted process graph.

Adding an action. Let $(bs_{init},a_1,bs_2,...,bs_k)$ be any feasible path after the addition of *a* to the set of actions. If *a* is not among the actions $a_1,a_2,...$ of the aforementioned path, this feasible path is contained in the given process graph. As our approach retains all paths from the given process graph, this path is contained in the adapted process graph as well. Now let *a* be contained in the actions $a_1,a_2,...$ of the selected feasible path. Then there is a belief state *bs* preceding (the first occurrence of) *a* in this path with $a\in app(bs)$. As elaborated in the design of our approach, we examine the path $(bs_{init},a_1,bs_2,...,bs)$ as it ends with a state in which *a* is applicable. From here, we conduct planning steps to determine and retain all feasible paths that extend $(bs_{init},a_1,bs_2,...,bs)$ and hence $(bs_{init},a_1,bs_2,...,bs_k)$ is contained in the adapted process graph.

Removing an action. Let $(bs_{init},a_1,bs_2,...,bs_k)$ be any feasible path after the removal of *a* from the set of actions *A*. As, in regard to Definition 5, actions must be part of the set of actions to be contained in a path, this path does not contain a. It is contained in the given process graph and, as such a path is not changed at all, retained in the adapted process graph according to our approach.

Updating an action. Strengthening update of the preconditions. Let $(bs_{init},a_1,bs_2,...,bs_k)$ be any feasible path after a strengthening update of the preconditions of an action a resulting in the action a'. If this path does not contain a', it remains unchanged, is retained from the given process graph and hence contained in the adapted process graph. If, however, the path contains a' at some place a_i (let i be the smallest index such that $a'=a_i$), a part of the considered path, more precisely $(bs_{init},a_1,bs_2,...,bs_i)$, is contained in the given process graph. From here, we proceed as in the case of updating the initial state and retrieve all feasible paths from bs_i. As this was proven to be complete, the feasible path $(bs_{init},a_1,bs_2,...,bs_k)$ is retrieved as well.

Updating an action. Weakening update of the preconditions. Let $(bs_{init},a_1,bs_2,...,bs_k)$ be any feasible path after a weakening update of the preconditions of an action a resulting in the action a'. If this path does not contain a', it remains unchanged, is retained from the given process graph and hence contained in the adapted process graph. If, however, the path contains a' at some place a_i (let i be the smallest index such that $a'=a_i$), it is either retrieved by conducting planning steps from a belief state bs_j with $j\leq i$ of the given process graph or by following the approach for the initial state (which, as seen above, is complete).

Updating the effects. Let $(bs_{init},a_1,bs_2,...,bs_k)$ be any feasible path after updating the effects of an action a resulting in the action a'. Again, if this path does not contain the updated action, the path is contained in the given process graph and, according to the approach, retained. Otherwise, there exists an index $i<k$ with $a'=a_i$ and $a'\neq a_j$ for all $j<i$. Here, we treat $R(bs_i,a')$ as the update of $R(bs_i,a)$ and follow the approach for an update of the initial state which retrieves all feasible paths (seen in the proof above) starting with $(bs_{init},a_1,bs_2,..., bs_i,a',R(bs_i,a'))$, $(bs_{init},a_1,bs_2,...,bs_k)$ being amongst them.

Thus, Theorem 2 is shown for each case and hence proven. *q.e.d.*

THEOREM 3. *The approach terminates.*

PROOF. To show that the algorithm terminates, we distinguish the possible changes as described in the approach.

Updating the initial state. At first, we will show that the traversal of each belief state terminates. The traversal of an old state immediately terminates as we retain the corresponding subgraph. Traversing an updated state bs, we first note that the computation of the set $app(bs)$ comes to an end and that the set $app(bs)$ is finite. This stems from the fact that checking the applicability of an action a (i.e., $\forall w\in pre(a)\ \exists u\in bs : v(w)=v(u) \wedge r(w)\cap r(u) \neq \emptyset$) is a comparison of the restrictions of a finite number of belief state tuples and hence terminates and the fact that we are provided with a finite set of actions. As a next step, the finite set of all following belief states is retrieved by executing the state transition function on each action in $app(bs)$ which terminates as an operation on finite sets of belief state tuples. To check which of these belief states are old, new or updated, we need to compare each one of them with each belief state of the given process graph, which is a finite graph. Thus, handling an updated state terminates. Dealing with a new belief state bs terminates in a similar way as we again have to compute $app(bs)$, the belief states following bs and classify them as old, new or updated. To sum up, each traversal step terminates. Hence, it suffices to show that the number of traversal steps is finite which follows from the fact that the number of distinct belief states one can reach from an initial state by combining and conducting actions from a finite set of actions is finite.

Adding a goal state. The case of adding a goal state $goal$ to the set of goal states $GOALS$ is handled by a depth-first search through the belief states of the given process graph. Since the number of these belief states is finite (cf. Definition 5), it suffices to show that the traversal of each belief state terminates. Thus, we examine the traversal of a belief state bs. As a first step, we check whether bs meets $goal$ (i.e., $\forall p\in goal$: $\exists p'\in bs$, $v(p)=v(p')$, $r(p')\subset r(p)$), which is a comparison of the restrictions of a finite number of belief state tuples and hence terminates. If bs indeed meets $goal$, the traversal of bs ends. Otherwise, we try to reach a belief state meeting $goal$ by conducting planning steps, which terminates as well.

Removing a goal state. For the finite number of paths of the given process graph which end at a belief state meeting $goal$, it is checked whether they can be extended so that they lead to one of the remaining goal states. This is done by conducting planning steps for a finite number of belief states, which terminates.

Updating a goal state. Strengthening update. For the finite number of paths of the given process graph which end at a belief state meeting $goal$, it is checked whether they can be extended so that they lead to one of goal states from $GOALS'=(GOALS\setminus\{goal\})\cup\{goal'\}$. This is done by conducting planning steps for a finite number of belief states, which terminates.

Updating a goal state. Weakening update. The case of updating a goal state *goal* to a goal state *goal'* which weakens the conditions of *goal* is handled by a depth-first search through the belief states of the given process graph. Since the number of these belief states is finite (cf. Definition 5), it suffices to show that the traversal of each belief state terminates. Thus, we examine the traversal of a belief state *bs*. As a first step, we check whether *bs* meets *goal'* (i.e., $\forall p\in goal'$: $\exists p'\in bs$, $v(p)=v(p')$, $r(p')\subset r(p)$) which is a comparison of the restrictions of a finite number of belief state tuples and hence terminates. If *bs* indeed meets *goal'*, the traversal of *bs* ends. Otherwise we try to reach a belief state meeting *goal'* by conducting planning steps, which terminates as well.

Adding an action. Adding an action *a* to the set of actions is handled by a depth-first search through the belief states of the given process graph. From each such belief state, planning steps are conducted in order to find belief states in which *a* is applicable and hence new feasible paths can possibly be constructed. As the conduction of planning steps terminates and the given process graph has a finite number of belief states (cf. Definition 5), the depth-first search terminates as well.

Removing an action. When removing an action *a* from *A* so that $A'=A\setminus\{a\}$, the finite set of all feasible paths of the given process graph is traversed. It is checked whether such a feasible path contains the action *a* in order to determine whether this path is retained. These checks terminate as each feasible path contains only a finite number of actions and thus our approach terminates.

Updating an action. Strengthening update of the preconditions. Again, all feasible paths of the given process graph are traversed in order to check whether in the belief states in which *a* was applicable, the updated action *a'* is applicable as well. If this is not the case, the path is not considered in the adapted process graph and the traversal of this path ends. On the other hand, if *a'* is applicable in a belief state *bs* with $a\in app(bs)$, it is checked whether $R(bs,a)$ and $R(bs,a')$ coincide, which requires a finite amount of set comparisons and hence terminates. Finally, if these belief states do not coincide, we follow the approach for updating the initial state, which, as seen above, terminates. As the given process graph contains a finite number of feasible paths, this means that our approach addressing the strengthening update of the preconditions of an action terminates.

Updating an action. Weakening update of the preconditions. This case is handled by a depth-first search through the belief states of the given process graph. From each belief state, planning steps are conducted in order to find belief states *bs* with $a\notin app(bs)$ and $a'\in app(bs)$ and possibly construct new feasible paths containing *a'*, which terminates. Additionally, there may be belief states *bs* in which both *a* and *a'* are applicable. In this case, it is checked whether $R(bs,a)$ and $R(bs,a')$ coincide, which requires a finite amount of set comparisons and hence terminates. If these belief states do not coincide, we apply our approach for updating the initial state, which, as seen above, terminates. As the given process graph has a finite number of belief states (cf. Definition 5), the depth-first search terminates as well.

Updating the effects. When updating the effects of an action *a* resulting in the action *a'*, we traverse each belief state of the given process graph in which *a* is applicable. As by Definition 5 the given process graph contains a finite number of belief states, the set of such belief states is finite as well. For each such belief state *bs*, we treat $R(bs,a')$ as the update of $R(bs,a)$ and apply our approach for updating the initial state, which terminates as seen above.

Thus, Theorem 3 is shown for each case and hence proven. *q.e.d.*

B Pseudocode of the Presented Approach

ALGORITHM 1: Pseudocode of the Presented Approach

```
def updateinit(updated_inital_state, original_inital_state):
    handleUpdatedState(updated_inital_state, original_inital_state)

def handleUpdatedState(updated_state, original_state):
    if checkForGoal(updated_state):
        return

    old_actions = originalModel.getFollowingActions(updated_state)
    new_actions = actionLibrary - old_actions
    for action in old_actions:
        if (action.preconditions.containsVariable(updated_belief_state_tuple) and
            isApplicable(action, updated_belief_state_tuple)) or
            not action.preconditions.containsVariable(updated_belief_state_tuple):
                old_following_state = originalModel.getFollowingState(original_state, action)
                new_following_state = old_following_state.update(updated_belief_state_tuple)
                adaptedModel.addTransition(updated_state, action, new_following_state)
                if originalModel.contains(new_following_state):
                    adaptedModel.addAll(originalModel.getSubgraphFromState(new_following_state))
                else:
                    handleUpdatedState(new_following_state)

    for action in new_actions:
        if action.preconditions.containsVariable(updated_belief_state_tuple) and
            isApplicable(action, updated_belief_state_tuple):
            following_state = apply(action, updated_state)
            adaptedModel.addTransition(updated_state, action, following_state)
            if originalModel.contains(following_state):
                adaptedModel.addAll(originalModel.getSubgraphFromState(following_state))
            elif isUpdatedState(following_state):
                handleUpdatedState(following_state)
            else:
                handleNewState(following_state)
```

```
def handleNewState(new_state):
    if checkForGoal(new_state):
        return

    following_states = planStateTransitions(getApplicableActions(new_state), new_state)
    for following_state in following_states:
        if originalModel.contains(following_state):
            adaptedModel.addAll(originalModel.getSubgraphFromState(following_state))
        elif isUpdatedState(following_state):
            handleUpdatedState(following_state)
        else:
            handleNewState(following_state)

def addGoal(new_goal_state):
    adaptedModel.addGoal(new_goal_state)
    for state in originalModel.states:
        if checkForParticularGoal(state, new_goal_state):
            adaptedModel.removeTransitions(originalModel.getSubgraphFromState(state))

    for state in unplannedStates(originalModel):
        if checkForParticularGoal(state, new_goal_state):

    adaptedModel.addTransitions(originalModel.getTransitionsFromTo(inital_state, state))

def removeGoal(old_goal_state):
    adaptedModel.removeGoal(old_goal_state)
    for state in pathEndingStates:
        if checkForParticularGoal(state, old_goal_state):
            planSubGraphFromState(adaptedModel, state)

    adaptedModel.removeTransitionsNotLeadingToGoalStates()

def updateGoal(updated_goal_state, original_goal_state):
    if isStrengtheningUpdate(updated_goal_state, original_goal_state):
        adaptedModel.addGoal(new_goal_state)
        removeGoal(original_goal_state)
    elif isWeakeningUpdate(updated_goal_state, original_goal_state):
        adaptedModel.removeGoal(original_goal_state)
        addGoal(updated_goal_state)
    else:
        addGoal(updated_goal_state)
        removeGoal(original_goal_state)
```

```
def addAction(new_action):
    actionLibrary.add(new_action)
    for state in originalModel.states:
        if isApplicable(new_action, state):
            following_state = apply(new_action, state)
            adaptedModel.addTransition(state, new_action, following_state)
            planSubGraphFromState(adaptedModel, following_state)

    for state in unplannedStates(originalModel):
        if isApplicable(new_action, state):

    adaptedModel.addTransitions(originalModel.getTransitionsFromTo(inital_state, state))
            following_state = apply(new_action, state)
            adaptedModel.addTransition(state, new_action, following_state)
            planSubGraphFromState(adaptedModel, following_state)

def removeAction(old_action):
    actionLibrary.remove(old_action)
    for transition in adaptedModel.stateTransitions:
        if old_action in transition:
            adaptedModel.removeTransition(transition)

    adaptedModel.removeTransitionsNotLeadingToGoalStates()

def updateAction(updated_action, original_action):
    actionLibrary.remove(original_action)
    actionLibrary.add(updated_action)
    if updated_action.preconditions != original_action.preconditions:
        if isStrengtheningUpdate(updated_action.preconditions,original_action.preconditions):
            strengtheningUpdatePreconditions(adaptedModel, updated_action, original_action)
        elif isWeakeningUpdate(updated_action.preconditions,original_action.preconditions):
            weakeningUpdatePreconditions(adaptedModel, updated_action, original_action)
        else:
            weakeningUpdatePreconditions(adaptedModel, updated_action, original_action)
            strengtheningUpdatePreconditions(adaptedModel, updated_action, original_action)

    if updated_action.effects != original_action.effects:
        for transition in adaptedModel.stateTransitions:
            if original_action in transition:
                replaceTransitionAndUpdate(transition, updated_action)

def strengtheningUpdatePreconditions(adaptedModel, updated_action, original_action):
    for transition in adaptedModel.stateTransitions:
        if original_action in transition:
            if isApplicable(updated_action, transition.fromState()):
                replaceTransitionAndUpdate(transition, updated_action)
            else:
                adaptedModel.removeTransition(transition)
                adaptedModel.removeTransitionsNotLeadingToGoalStates()
```

```
def weakeningUpdatePreconditions(adaptedModel, updated_action, original_action):
    for state in originalModel.states:
        if isApplicable(original_action, state):
            transition = adaptedModel.findTransition(state, original_action)
            replaceTransitionAndUpdate(transition, updated_action)
        elif isApplicable(updated_action, state):
            following_state = apply(updated_action, state)
            adaptedModel.addTransition(state, updated_action, following_state)
            planSubGraphFromState(adaptedModel, following_state)

    for state in unplannedStates(originalModel):
        if isApplicable(updated_action, state):

    adaptedModel.addTransitions(originalModel.getTransitionsFromTo(inital_state, state))
            following_state = apply(updated_action, state)
            adaptedModel.addTransition(state, updated_action, following_state)
            planSubGraphFromState(adaptedModel, following_state)

def replaceTransitionAndUpdate(transition, updated_action):
    adaptedModel.removeTransition(transition)
    old_following_state = transition.fromState()
    updated_belief_state_tuple = applyForUpdatedBeliefState(updated_action, old_following_state)
    new_following_state = old_following_state.update(updated_belief_state_tuple)
    adaptedModel.addTransition(transition.fromState(), updated_action, new_following_state)
    handleUpdatedState(new_following_state)
```

C Simulation Experiment (E4.2)

In order to evaluate the performance of our approach in comparison to planning from scratch, we conducted a simulation experiment and focus on atomic changes to provide transparent results. However, please note that all possible adaptations can be realized as a sequence of these atomic changes. For our analysis, we measured absolute runtimes as well as each ratio, which means, the absolute runtime for adaptation divided by the corresponding absolute runtime for planning from scratch for all possible types of atomic change (cf. Table 2). To do so, we used adaptation cases based upon 12 existing real-world process graphs of different companies from the application contexts *Project Management*, *Insurance Management*, *Loan Management* and *Private Banking* (cf. Table 8). The process graphs consist of 17 to 8,267 actions and 15 to 2,693 belief states and contain numeric domains as well as discrete domains in their belief state tuples. There are two process graphs that stand out regarding the number of feasible paths (*Selling an insurance contract* and *Contracting wealth management customer*). In these two cases, large parts could be conducted in parallel to other large parts in the same process. Process graphs do not contain control flow patterns and paths represent sequences of subsequent belief states and actions. Hence, these two process graphs represent large processes of complete value chains (including back office) in which many actions can be executed in multiple different orders. This results in process graphs comprising a vast number of feasible paths, each of which represents a different possible order of actions. Yet, the number of distinct state variables remains rather small because these central business state variables can have many different values and auxiliary state variables, which were not counted here. We deliberately analyzed these two process graphs to show the feasibility of our approach regarding large process graphs.

In a first step, we defined random adaptation cases for each type of atomic change based on the belief state variables of each of the 12 process graphs. Each of these generated adaptation cases was then (automatically) validated with regard to its validity. For instance, a goal state could only be removed if there was at least one goal state remaining after the adaptation. Invalid adaptation cases remained unconsidered. We randomly generated 1,500 valid adaptation cases for each type of atomic change over all 12 process graphs. For each case, the specification of the process graph, the specification of the adaptation and the specification for planning the adapted process graph from scratch were automatically prepared in terms of XML files that could be imported in our prototypical implementation.

We applied our approach to these adaptation cases and automatically verified that adapting the given process graphs and planning the adapted process graphs from scratch resulted in exactly the same process model in each case. Then, we compared the runtime required for adapting the given process graphs with the runtime required for planning the graphs from scratch. Both runtimes do not take into account the time required to generate the XML files representing the adaptation cases. The results of this runtime comparison can be seen in Figure 6.

We observed that the required runtime for adapting the existing process graphs by the prototype is lower than for planning the graphs from scratch for each type of atomic change. The left part of Figure 6 shows the mean of the required absolute runtime for adapting the process graphs (first line in each cell) as well as the mean percentage ratio (absolute runtime for adaptation divided by absolute runtime for planning from scratch; second line in each cell), which varies between 3.68% and 10.52%, depending on the type of atomic change. To give an example of a process, independent of the type of atomic change our approach takes on average 0.35 seconds for adapting the graph of the process *Selling an insurance contract*.

In contrast, planning from scratch would on average take about 10 seconds, which leads to an average time saving of 96.4%. The right part of Figure 6 shows box plots of these percentage ratios. The left and right ends of the boxes are the first and third quartiles, and the bands inside the boxes are the medians. The whiskers (i.e., the horizontal lines outside of the boxes) include all values within 1.5*interquartile range.

These results of the simulation experiment show that while for just a few adaptation cases, adapting instead of planning from scratch provided only negligible or even non-existent runtime advantages, for most adaptation cases, the runtime advantage was considerable. Further, the results support that using our approach provides considerable performance advantages, even if a sequence of changes has to be addressed. To analyze the significance of our findings, we further conducted a one-tailed paired t-test. This kind of test was chosen because we aimed to analyze a paired sample dataset (runtimes for adapting versus planning from scratch). There was a significant difference in the runtimes according to the results of the t-test: *degrees of freedom*=139,190; *t-value*=88.685, *p-value*=2.2e-16. The *p*-value was consistently less than or equal to 2.2e-16 across all types of atomic changes. These significant results support the thesis that our approach is faster than planning the adapted process graphs from scratch. This is especially advantageous when working with graphical process modeling tools. In such tools, modelers can – *in real time* – conduct a sequence of changes in order to, step by step, adapt a given process model to needs for change. The previously mentioned example of adapting *Selling an insurance contract* underlines this argument. Here, for instance, using the presented approach would on average result in a 0.35 second waiting time after each entered change instead of 10 seconds when planning from scratch. Thus, our approach enables modelers to work much more comfortably compared to using existing approaches.

	Add	**Update**	**Remove**
Initial state	---	7.73ms 9.05%	---
Goal states	3.52ms 3.80%	6.07ms 7.32%	0.58ms 3.68%
Actions	11.67ms 10.52%	18.68ms 7.80%	2.87ms 5.71%

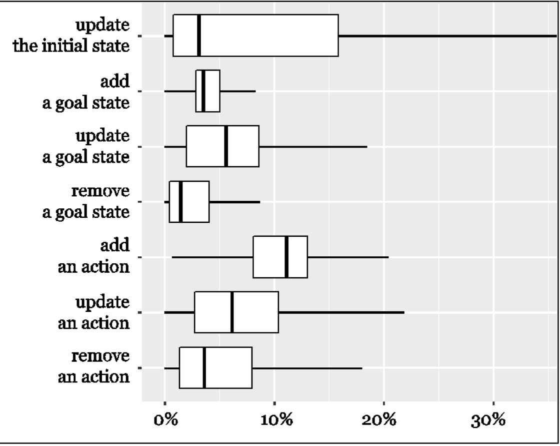

Fig. O1. Evaluation Results by means of a Prototypical Implementation.

Table O1. Key Properties of used real-world Processes

Application Context	**Process description**	**Number of actions in the given process graph**	**Number of belief states in the given process graph**	**Number of feasible paths in the given process graph**	**Number of distinct state variables**
Project Management	Preparing and coordinating project profile	17	15	6	4
Project Management	Specifying project resources	25	18	36	7
Project Management	Allocating project resources	26	22	24	6
Project Management	Preparing the project report for the board	38	25	252	5
Insurance Management	Administrating customer and product database	43	38	52	7
Insurance Management	Handling insured events	54	44	60	6
Insurance Management	Selling an insurance contract	8,267	2,693	97,501,324,491	9
Loan Management	Analyzing credit rating	40	31	197	6
Loan Management	Selling mortgage loans	57	43	216	8
Loan Management	Settling mortgage loans	122	69	6.572	11
Private Banking	Contracting wealth management customer	278	189	1,244,416	27
Private Banking	Order management	82	74	32	19