A Algorithms

Algorithm 1: Caching a skill in an abstract domain.

Algorithm `cachSkill`

Inputs:

A state trace from the skill execution $\mathfrak{S}(\boldsymbol{E})$

A public abstraction key $PubKey = \left(\texttt{project}_p, \texttt{reconst}_p, \mathcal{P}_s\right)$

Output:

Caching success flag

$validPrivateKeys \leftarrow \bigcap_{\boldsymbol{S} \in \mathfrak{S}(\boldsymbol{E})} \mathcal{P}_{\boldsymbol{S}}$

if $validPrivateKeys \neq \emptyset$ **then**

Select a key p from $validPrivateKeys$

$abstractTrace \leftarrow \texttt{project}_p\left(\mathfrak{S}(\boldsymbol{E})\right)$

$DataBase$.`add`($abstractTrace$, $PubKey$)

return True `// caching succeeded`

end

return False `// caching failed, try another public key`

Algorithm 2: Get a private key for reconstruction (applicability test).

Algorithm `getPrivateKey`

Inputs:

An abstract skill's state trace $(\boldsymbol{\xi}_0, \ldots, \boldsymbol{\xi}_n)$

A start state $\boldsymbol{S}_{\text{start}}$

A task goal state $\boldsymbol{S}_{\text{goal}}$

A public key $PubKey = \left(\texttt{project}_p, \texttt{reconst}_p, \mathcal{P}_s\right)$

Output:

If the abstract skill is applicable, returns a private key for reconstruction; otherwise, returns **False**

if $\left|\mathcal{P}^{\boldsymbol{\xi}_0}_{\boldsymbol{S}_{start}}\right| = 1$ **then**

  $p \leftarrow \mathcal{P}^{\boldsymbol{\xi}_0}_{\boldsymbol{S}_{\text{start}}}$

  if $\texttt{project}_p\left(\boldsymbol{S}_{goal}\right) = \boldsymbol{\xi}_n$ *or* $\texttt{reconst}_p\left(\boldsymbol{\xi}_n\right) = \boldsymbol{S}_{goal}$ **then**

    return p `// applicable`

  end

end

else if $\left|\mathcal{P}^{\boldsymbol{\xi}_0}_{\boldsymbol{S}_{start}}\right| > 1$ **then**

  $intersection \leftarrow \mathcal{P}^{\boldsymbol{\xi}_n}_{\boldsymbol{S}_{\text{goal}}} \cap \mathcal{P}^{\boldsymbol{\xi}_0}_{\boldsymbol{S}_{\text{start}}}$

  if $intersection \neq \emptyset$ **then**

    Select a key p from $intersection$

    return p `// applicable`

  end

end

return False `// skill inapplicable`

Algorithm 3: Reconstruction of a cached abstract skill.

Algorithm `reconstructSkill`

Inputs:
An abstract skill's state trace $(\boldsymbol{\xi}_0, \ldots, \boldsymbol{\xi}_n)$
A public key $PubKey = \left(\texttt{project}_p, \texttt{reconst}_p, \mathcal{P}_s\right)$
A "classical" task planning problem $P \doteq (D,\, \boldsymbol{S}_{\text{start}},\, \boldsymbol{S}_{\text{goal}})$

Output:
A plan that solves the given problem, if reconstruction succeeded; otherwise, returns **False**

$p \leftarrow$ `getPrivateKey`$(\boldsymbol{\xi}_0, \boldsymbol{\xi}_n, \boldsymbol{S}_{start}, \boldsymbol{S}_{goal}, PubKey)$
if $p \neq$ **False then** `// skill is applicable`
  $states \leftarrow \texttt{reconst}_p\,(\boldsymbol{\xi}_0, \ldots, \boldsymbol{\xi}_n)$ `// reconstruct state trace`
  $actions \leftarrow$ `recoverActions`$(states, D)$ `// recover actions`
  if $actions \neq$ **False then**
    return $actions$
  end
end
return False

Procedure `recoverActions`

Inputs:
A state trace $(\boldsymbol{S}_0, \ldots, \boldsymbol{S}_n)$
A planning domain D

Output:
If feasible, returns a sequence of actions that follows the states in the trace; otherwise, returns **False**

for $i \in \{1, \ldots, n\}$ **do**
  if $\exists$ *an action* $a \in \mathcal{A}$ *between* $\boldsymbol{S}_{i-1}$ *and* $\boldsymbol{S}_i$ **then**
    $a_i \leftarrow a$
  end
  else
    return False `// unfeasible`
  end
end
return $(a_1, \ldots, a_n)$ `// feasible`