# Verified SAT-Based AI Planning 

Mohammad Abdulaziz and Friedrich Kurz*

We present an executable formally verified SAT encoding of classical AI planning that is based on the encodings by Kautz and Selman [2] and the one by Rintanen et al. [3]. The encoding was experimentally tested and shown to be usable for reasonably sized standard AI planning benchmarks. We also use it as a reference to test a state-of-the-art SAT-based planner, showing that it sometimes falsely claims that problems have no solutions of certain lengths. The formalisation in this submission was described in an independent publication [1].

## Contents

1 State-Variable Representation ..... 3
2 STRIPS Representation ..... 3
3 STRIPS Semantics ..... 5
3.1 Serial Plan Execution Semantics ..... 5
3.2 Parallel Plan Semantics ..... 15
3.3 Serializable Parallel Plans ..... 46
3.4 Auxiliary lemmas about STRIPS ..... 58
4 SAS+ Representation ..... 58
5 SAS+ Semantics ..... 64
5.1 Serial Execution Semantics ..... 64
5.2 Parallel Execution Semantics ..... 66
5.3 Serializable Parallel Plans ..... 77
5.4 Auxiliary lemmata on SAS+ ..... 83
6 SAS+/STRIPS Equivalence ..... 85
6.1 Translation of SAS+ Problems to STRIPS Problems ..... 85
6.2 Equivalence of SAS+ and STRIPS ..... 166

[^0]7 The Basic SATPlan Encoding ..... 177
7.1 Encoding Function Definitions ..... 177
7.2 Decoding Function Definitions ..... 180
7.3 Soundness of the Basic SATPlan Algorithm ..... 212
7.4 Completeness ..... 250
8 Serializable SATPlan Encodings ..... 291
8.1 Soundness ..... 298
8.2 Completeness ..... 301
9 SAT-Solving of SAS+ Problems ..... 308
10 Adding Noop actions to the SAS+ problem ..... 310
11 Proving Equivalence of SAS+ representation and Fast-Downward's Multi-Valued Problem Representation ..... 312
11.1 Translating Fast-Downward's represnetation to SAS+ ..... 312
11.2 Translating SAS+ represnetation to Fast-Downward's ..... 328
11.3 SAT encoding works for Fast-Downward's representation ..... 334
12 DIMACS-like semantics for CNF formulae ..... 334
12.1 Going from Formualae to DIMACS-like CNF ..... 340
13 Code Generation ..... 346

```
theory State-Variable-Representation
    imports Main Propositional-Proof-Systems.Formulas Propositional-Proof-Systems.Sema
    Propositional-Proof-Systems.CNF
begin
```


## 1 State-Variable Representation

Moving on to the Isabelle implementation of state-variable representation, we first add a more concrete representation of states using Isabelle maps. To this end, we add a type synonym for maps of variables to values. Since maps can be conveniently constructed from lists of assignments-i.e. pairs $(v, a)::$ 'variable $\times$ 'domain-we also add a corresponding type synonym .
type-synonym ('variable, 'domain) state $=$ 'variable $\rightharpoonup$ 'domain
type-synonym ('variable, 'domain) assignment $=$ 'variable $\times$ 'domain
Effects and effect condition (see ??) are implemented in a straight forward manner using a datatype with constructors for each effect type

```
type-synonym('variable, 'domain) Effect = ('variable }\times\mathrm{ 'domain) list
end
```

theory STRIPS-Representation
imports State-Variable-Representation
begin

## 2 STRIPS Representation

We start by declaring a record for STRIPS operators. This which allows us to define a data type and automatically generated selector operations. ${ }^{1}$ The record specification given below closely resembles the canonical representation of STRIPS operators with fields corresponding to precondition, add effects as well as delete effects.

```
record ('variable) strips-operator \(=\)
    precondition-of :: 'variable list
    add-effects-of :: 'variable list
    delete-effects-of :: 'variable list
```

- This constructor function is sometimes a more descriptive and replacement for the record syntax and can moreover be helpful if the record syntax leads to type ambiguity.

[^1]
## abbreviation operator-for

:: 'variable list $\Rightarrow$ 'variable list $\Rightarrow$ 'variable list $\Rightarrow$ 'variable strips-operator where operator-for pre add delete $\equiv 0$

$$
\text { precondition-of }=\text { pre }
$$

, add-effects-of $=$ add
, delete-effects-of $=$ delete D

## definition to-precondition

:: 'variable strips-operator $\Rightarrow$ ('variable, bool) assignment list
where to-precondition $o p \equiv \operatorname{map}(\lambda v .(v$, True $))($ precondition-of op)
definition to-effect
:: 'variable strips-operator $\Rightarrow$ ('variable, bool) Effect
where to-effect $o p=\left[\left(v_{a}\right.\right.$, True $) . v_{a} \leftarrow a d d$-effects-of $\left.o p\right] @\left[\left(v_{d}\right.\right.$, False $) . v_{d} \leftarrow$ delete-effects-of op]

Similar to the operator definition, we use a record to represent STRIPS problems and specify fields for the variables, operators, as well as the initial and goal state.
record ('variable) strips-problem $=$
variables-of :: 'variable list ((-v) [1000] 999)
operators-of :: 'variable strips-operator list ((-○) [1000] 999)
initial-of :: 'variable strips-state ((-I) [1000] 999)
goal-of :: 'variable strips-state ( $(-G)$ [1000] 999)
value stop
As discussed in ??, the effect of a STRIPS operator can be normalized to a conjunction of atomic effects. We can therefore construct the successor state by simply converting the list of add effects to assignments to True resp. converting the list of delete effect to a list of assignments to False and then adding the map corresponding to the assignments to the given state $s$ as shown below in definition ??. ${ }^{2}$

```
definition execute-operator
    :: 'variable strips-state
    \(\Rightarrow\) 'variable strips-operator
    \(\Rightarrow\) 'variable strips-state (infixl \(\gg 52\) )
    where execute-operator s op
    \(\equiv s++\) map-of (effect-to-assignments op)
end
theory STRIPS-Semantics
    imports STRIPS-Representation
        List-Supplement
    Map-Supplement
begin
```

[^2]
## 3 STRIPS Semantics

Having provided a concrete implementation of STRIPS and a corresponding locale strips, we can now continue to define the semantics of serial and parallel STRIPS plan execution (see ?? and ??).

### 3.1 Serial Plan Execution Semantics

Serial plan execution is defined by primitive recursion on the plan. Definition ?? returns the given state if the state argument does note satisfy the precondition of the next operator in the plan. Otherwise it executes the rest of the plan on the successor state $s \gg o p$ of the given state and operator.

```
primrec execute-serial-plan
    where execute-serial-plan s[] = s
    execute-serial-plan s (op # ops)
        =(if is-operator-applicable-in s op
            then execute-serial-plan (execute-operator s op) ops
            else s
    )
```

Analogously, a STRIPS trace either returns the singleton list containing only the given state in case the precondition of the next operator in the plan is not satisfied. Otherwise, the given state is prepended to trace of the rest of the plan for the successor state of executing the next operator on the given state.

```
fun trace-serial-plan-strips
    :: 'variable strips-state }=>\mathrm{ 'variable strips-plan }=>\mathrm{ 'variable strips-state list
    where trace-serial-plan-strips s [] = [s]
    | trace-serial-plan-strips s (op # ops)
        =s# (if is-operator-applicable-in s op
            then trace-serial-plan-strips (execute-operator s op) ops
            else [])
```

Finally, a serial solution is a plan which transforms a given problems initial state into its goal state and for which all operators are elements of the problem's operator list.

```
definition is-serial-solution-for-problem
    where is-serial-solution-for-problem \(\Pi \pi\)
        \(\equiv(\) goal-of \(\Pi) \subseteq_{m}\) execute-serial-plan (initial-of \(\Pi\) ) \(\pi\)
            \(\wedge\) list-all ( \(\lambda\) op. ListMem op (operators-of \(\Pi\) ) ) \(\pi\)
lemma is-valid-problem-strips-initial-of-dom:
    fixes П:: 'a strips-problem
    assumes is-valid-problem-strips \(\Pi\)
    shows \(\operatorname{dom}\left((\Pi)_{I}\right)=\operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\)
    proof -
```

```
    {
        let ?I = strips-problem.initial-of \Pi
        let ?vs = strips-problem.variables-of \Pi
        fix v
        have ?I v\not= None \longleftrightarrowListMem v ?vs
        using assms(1)
        unfolding is-valid-problem-strips-def
        by meson
    hence v\in dom?I \longleftrightarrowv\in set ?vs
        using ListMem-iff
        by fast
    }
    thus ?thesis
        by auto
    qed
lemma is-valid-problem-dom-of-goal-state-is:
    fixes П:: 'a strips-problem
    assumes is-valid-problem-strips \Pi
    shows dom }((\Pi\mp@subsup{)}{G}{})\subseteq\operatorname{set}((\Pi)\mathcal{V}
    proof -
        let ?vs = strips-problem.variables-of \Pi
        let ?G = strips-problem.goal-of \Pi
        have nb:\forallv. ?G v\not= None \longrightarrowListMem v ?vs
        using assms(1)
        unfolding is-valid-problem-strips-def
        by meson
    {
        fix v
        assume v \in dom?G
        then have ?G v}\not=\mathrm{ None
            by blast
        hence v\in set ?vs
            using nb
            unfolding ListMem-iff
            by blast
    }
    thus ?thesis
            by auto
qed
lemma is-valid-problem-strips-operator-variable-sets:
    fixes П:: 'a strips-problem
    assumes is-valid-problem-strips \Pi
        and op }\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{}
    shows set (precondition-of op) \subseteq set ((\Pi)\mathcal{V})
        and set (add-effects-of op)\subseteq set ((\Pi)\mathcal{V})
        and set (delete-effects-of op) \subseteq set ((\Pi)v)
        and disjnt (set (add-effects-of op)) (set (delete-effects-of op))
```

```
proof -
    let ?ops = strips-problem.operators-of \Pi
        and ?vs = strips-problem.variables-of \Pi
    have list-all (is-valid-operator-strips \Pi) ?ops
        using assms(1)
        unfolding is-valid-problem-strips-def
        by meson
    moreover have }\forallv\in\operatorname{set}(precondition-of op).v\in set ((\Pi)\mathcal{V}
        and }\forallv\in\operatorname{set (add-effects-of op).v\in set ((\Pi)\mathcal{V})
        and }\forallv\in\operatorname{set}(delete-effects-of op).v\in set ((\Pi)\mathcal{V}
        and }\forallv\in\mathrm{ set (add-effects-of op).v}\not\in set (delete-effects-of op
        and }\forallv\in\mathrm{ set (delete-effects-of op).v & set (add-effects-of op)
        using assms(2) calculation
        unfolding is-valid-operator-strips-def list-all-iff Let-def ListMem-iff
        using variables-of-def
        by auto+
    ultimately show set (precondition-of op) \subseteqset ((\Pi)\mathcal{V})
        and set (add-effects-of op)\subseteq set ((\Pi)v)
        and set (delete-effects-of op)}\subseteq\mathrm{ set ((П)v)
        and disjnt (set (add-effects-of op)) (set (delete-effects-of op))
        unfolding disjnt-def
        by fast+
qed
lemma effect-to-assignments-i:
assumes as = effect-to-assignments op
shows as = (map (\lambdav. (v, True)) (add-effects-of op)
    @ map (\lambdav. (v, False)) (delete-effects-of op))
using assms
unfolding effect-to-assignments-def effect--strips-def
by auto
```

lemma effect-to-assignments-ii:

- NOTE effect-to-assignments can be simplified drastically given that only atomic effects and the add-effects as well as delete-effects lists only consist of variables.
assumes as =effect-to-assignments op
obtains $a s_{1} a s_{2}$
where $a s=a s_{1} @ a s_{2}$
and $a s_{1}=\operatorname{map}(\lambda v .(v$, True $))($ add-effects-of op $)$
and $a s_{2}=\operatorname{map}(\lambda v .(v$, False $))($ delete-effects-of op $)$
by (simp add: assms effect--strips-def effect-to-assignments-def)
- NOTE Show that for every variable $v$ in either the add effect list or the delete effect list, there exists an assignment in representing setting $v$ to true respectively setting $v$ to false. Note that the first assumption amounts to saying that the add effect list is not empty. This also requires us to split lemma into two separate lemmas since add and delete effect lists are not required to both contain at least one variable simultaneously.
lemma effect-to-assignments-iii-a:

```
fixes v
assumes v\in set (add-effects-of op)
    and as = effect-to-assignments op
obtains a where a\in set as a=(v, True)
proof -
    let ?add-assignments = (\lambdav.(v, True))' set (add-effects-of op)
    let ?delete-assignments = (\lambdav.(v, False))'set (delete-effects-of op)
    obtain ass as 
        where a1: as =as1@ @ as2
            and a2:as }=\mathrm{ map ( }\lambdav.(v,\mathrm{ True)) (add-effects-of op)
            and a3:as 2 = map (\lambdav. (v, False)) (delete-effects-of op)
        using assms(2) effect-to-assignments-ii
        by blast
    then have b: set as
        =?add-assignments \cup ?delete-assignments
        by auto
```

    - NOTE The existence of an assignment as proposed can be shown by the
    following sequence of set inclusions.
\{
from $b$ have ?add-assignments $\subseteq$ set as
by blast
moreover have $\{(v$, True $)\} \subseteq$ ?add-assignments
using assms(1) a2
by blast
ultimately have $\exists a . a \in$ set as $\wedge a=(v$, True $)$
by blast
\}
then show ?thesis
using that
by blast
qed
lemma effect-to-assignments-iii-b:

- NOTE This proof is symmetrical to the one above.
fixes $v$
assumes $v \in$ set (delete-effects-of op)
and as $=$ effect-to-assignments op
obtains $a$ where $a \in$ set as $a=(v$, False)
proof -
let ?add-assignments $=(\lambda v .(v$, True $))$ ' set (add-effects-of op)
let ?delete-assignments $=(\lambda v .(v$, False $))$ 'set (delete-effects-of op)
obtain $a s_{1} a s_{2}$
where a1: as $=a s_{1} @ a s_{2}$
and $a 2: a s_{1}=\operatorname{map}(\lambda v .(v$, True $))($ add-effects-of op)
and $a 3: a s_{2}=\operatorname{map}(\lambda v .(v$, False $))($ delete-effects-of op)
using assms(2) effect-to-assignments-ii
by blast
then have $b$ : set as
$=$ ?add-assignments $\cup$ ?delete-assignments
by auto
- NOTE The existence of an assignment as proposed can be shown by the following sequence of set inclusions.

```
    {
        from b have ?delete-assignments }\subseteq\mathrm{ set as
            by blast
```



```
            using assms(1) a2
            by blast
            ultimately have }\existsa.a\in\mathrm{ set as ^ a = (v, False)
            by blast
    }
    then show ?thesis
        using that
        by blast
    qed
```

lemma effect--strips- $i$ :
fixes $o p$
assumes $e=$ effect--strips op
obtains $e s_{1} e s_{2}$
where $e=\left(e s_{1} @ e s_{2}\right)$
and $e s_{1}=\operatorname{map}(\lambda v .(v$, True $))($ add-effects-of op $)$
and $e s_{2}=\operatorname{map}(\lambda v .(v$, False $))($ delete-effects-of op $)$
proof -
obtain $e s_{1} e s_{2}$ where $a: e=\left(e s_{1} @ e s_{2}\right)$
and $b: e s_{1}=\operatorname{map}(\lambda v .(v$, True $))($ add-effects-of op)
and $c: e s_{2}=\operatorname{map}(\lambda v .(v$, False $))($ delete-effects-of op $)$
using assms(1)
unfolding effect--strips-def
by blast
then show?thesis
using that
by force
qed
lemma effect--strips-ii:
fixes $o p$
assumes $e=$ ConjunctiveEffect (es ${ }_{1}$ @ es $s_{2}$ )
and $e s_{1}=\operatorname{map}(\lambda v .(v$, True $))($ add-effects-of op $)$
and $e s_{2}=\operatorname{map}(\lambda v .(v$, False $))($ delete-effects-of op $)$
shows $\forall v \in \operatorname{set}(a d d$-effects-of op $)$. $\left(\exists e^{\prime} \in \operatorname{set} e s_{1} . e^{\prime}=(v\right.$, True $\left.)\right)$
and $\forall v \in$ set (delete-effects-of op $).\left(\exists e^{\prime} \in\right.$ set es $2 . e^{\prime}=(v$, False $\left.)\right)$
proof

- NOTE Show that for each variable $v$ in the add effect list, we can obtain an atomic effect with true value.
fix $v$
\{
assume $a: v \in \operatorname{set}$ (add-effects-of op)

```
    have set es 
            using assms(2) List.set-map
            by auto
    then obtain e}\mp@subsup{e}{}{\prime
        where e}\mp@subsup{e}{}{\prime}\in\mathrm{ set es,
        and \mp@subsup{e}{}{\prime}=(\lambdav.(v,True)) v
        using a
        by blast
        then have \exists 的\in set es.. e}=(v,\mathrm{ True )
        by blast
    }
    thus v\in set (add-effects-of op)\Longrightarrow\exists\mp@subsup{e}{}{\prime}\in\mathrm{ set es. }\mp@subsup{\mp@code{l}}{1}{}\cdot\mp@subsup{e}{}{\prime}=(v,\mathrm{ True)}
        by fast
    - NOTE the proof is symmetrical to the one above: for each variable v in the
delete effect list, we can obtain an atomic effect with v being false.
    next
    {
        fix v
        assume a:v\in set (delete-effects-of op)
        have set esz = (\lambdav. (v, False))'set (delete-effects-of op)
            using assms(3) List.set-map
            by force
        then obtain e"
            where e"' & set es2
            and }\mp@subsup{e}{}{\prime\prime}=(\lambdav.(v, False))
            using a
            by blast
        then have \exists}\mp@subsup{e}{}{\prime\prime}\in\mathrm{ set es2. . }\mp@subsup{e}{}{\prime\prime}=(v,\mathrm{ False)
            by blast
    }
    thus }\forallv\inset (delete-effects-of op). \existse'\inset es2. e' = (v, False
        by fast
    qed
lemma map-of-constant-assignments-dom:
- NOTE ancillary lemma used in the proof below.
assumes m= map-of (map ( }\lambdav.(v,d))vs
shows dom m = set vs
proof -
    let ?vs' = map (\lambdav. (v,d)) vs
    have dom m= fst ' set ?vs'
        using assms(1) dom-map-of-conv-image-fst
        by metis
    moreover have fst ' set ?vs' = set vs
        by force
    ultimately show ?thesis
        by argo
qed
```

```
lemma effect-strips-iii-a:
    assumes }\mp@subsup{s}{}{\prime}=(s>>op
    shows }\v.v\in\mathrm{ set (add-effects-of op) C s'v=Some True
    proof -
    fix v
    assume a:v\in set (add-effects-of op)
    let ?as= effect-to-assignments op
    obtain as as as where b: ?as=as = @ as 2
        and c:as s = map (\lambdav. (v, True)) (add-effects-of op)
        and as }\mp@subsup{\mp@code{2}}{2}{=map(\lambdav.(v, False))(delete-effects-of op)
        using effect-to-assignments-ii
        by blast
    have d: map-of ?as = map-of as }\mp@subsup{\mp@code{2}}{2}{++}\mathrm{ map-of as 
        using b Map.map-of-append
        by auto
    {
        - TODO refactor?
        let ?vs = add-effects-of op
        have ?vs \not= []
            using a
            by force
    then have dom (map-of as s) = set (add-effects-of op)
            using c map-of-constant-assignments-dom
            by metis
    then have v}\in\operatorname{dom}(map-of as )
            using a
            by blast
    then have map-of ?as v = map-of as1 v
            using d
            by force
    } moreover {
        let ?f = \lambda-. True
        from c have map-of as }\mp@subsup{s}{1}{}=(\mathrm{ Some O ?f) |'(set (add-effects-of op))
            using map-of-map-restrict
            by fast
        then have map-of as 
            using a
            by auto
    }
    moreover have s'=s++ map-of as 2 ++ map-of as 
        using assms(1)
        unfolding execute-operator-def
        using }
        by simp
    ultimately show s'v}=\mathrm{ Some True
        by simp
qed
```

```
lemma effect--strips-iii-b:
    assumes }\mp@subsup{s}{}{\prime}=(s>>op
    shows \v.v\in set (delete-effects-of op) ^v\notv set (add-effects-of op)\Longrightarrows'v=
Some False
    proof (auto)
    fix v
    assume a1:v & set (add-effects-of op) and a2:v set (delete-effects-of op)
    let ?as = effect-to-assignments op
    obtain as as as where b: ?as=as1 @ as 2
    and c:as_ = map (\lambdav. (v, True)) (add-effects-of op)
    and d:as⿱2 = map (\lambdav. (v, False))(delete-effects-of op)
    using effect-to-assignments-ii
    by blast
have e: map-of ?as = map-of as 2 ++ map-of as 
    using b Map.map-of-append
    by auto
{
    have dom(map-of as 
        using c map-of-constant-assignments-dom
        by metis
    then have v\not\indom (map-of as )
            using a1
    by blast
} note f}=\mathrm{ this
{
    let ?vs = delete-effects-of op
    have ?vs \not= []
            using a2
            by force
        then have dom (map-of as_) = set ?vs
            using d map-of-constant-assignments-dom
            by metis
} note g= this
{
have s}\mp@subsup{s}{}{\prime}=s++\mathrm{ map-of as }\mp@subsup{\mp@code{2}}{2}{++}\mathrm{ map-of as 
            using assms(1)
            unfolding execute-operator-def
            using b
            by simp
            thm f map-add-dom-app-simps(3)[OF f, of s++ map-of as }\mp@subsup{s}{2}{}
            moreover have s'v=(s++ map-of as 2) v
                using calculation map-add-dom-app-simps(3)[OF f, of s ++ map-of as 2]
            by blast
                            moreover have v\indom (map-of as⿱\mp@code{)}
            using a2 g
            by argo
            ultimately have }\mp@subsup{s}{}{\prime}v=map-of as\mp@subsup{s}{2}{}
            by fastforce
```

```
    }
    moreover
    {
        let ?f = \lambda-. False
        from d have map-of as 2 = (Some ○?f)|'(set (delete-effects-of op))
            using map-of-map-restrict
            by fast
        then have map-of as, v = Some False
        using a2
        by force
    }
    ultimately show s}\mp@subsup{s}{}{\prime}v=\mathrm{ Some False
        by argo
qed
lemma effect--strips-iii-c:
assumes }\mp@subsup{s}{}{\prime}=(s>>op
shows \v.v\not\in set (add-effects-of op) ^v\not\inset (delete-effects-of op) \Longrightarrows'v=
sv
proof (auto)
    fix v
    assume a1:v\not\in set (add-effects-of op) and a2:v v set (delete-effects-of op)
    let ?as= effect-to-assignments op
    obtain as }a\mp@subsup{s}{2}{}\mathrm{ where b: ?as=as \ @ as 2
        and c:as 的 = map (\lambdav. (v, True)) (add-effects-of op)
        and d:as⿱2 = map (\lambdav. (v, False))(delete-effects-of op)
        using effect-to-assignments-ii
        by blast
    have e: map-of ?as = map-of as 
        using b Map.map-of-append
        by auto
    {
    have dom(map-of as ) = set (add-effects-of op)
        using c map-of-constant-assignments-dom
        by metis
        then have v\not\indom (map-of as )
            using a1
            by blast
    } moreover {
    have dom (map-of as }\mp@subsup{\mp@code{S}}{2}{\prime})=\mathrm{ set (delete-effects-of op)
        using d map-of-constant-assignments-dom
        by metis
        then have v\not\indom (map-of as2)
            using a2
        by blast
    }
    ultimately show s}\mp@subsup{s}{}{\prime}v=s
        using assms(1)
```

```
    unfolding execute-operator-def
    by (simp add: b map-add-dom-app-simps(3))
qed
```

The following theorem combines three preceding sublemmas which show that the following properties hold for the successor state $s^{\prime} \equiv$ execute-operator $o p s$ obtained by executing an operator $o p$ in a state $s:{ }^{3}$

- every add effect is satisfied in $s^{\prime}$ (sublemma ); and,
- every delete effect that is not also an add effect is not satisfied in $s^{\prime}$ (sublemma ); and finally
- the state remains unchanged-i.e. $s^{\prime} v=s v$-for all variables which are neither an add effect nor a delete effect.

```
theorem operator-effect--strips:
    assumes \(s^{\prime}=(s \gg o p)\)
    shows
    \(\wedge v\).
        \(v \in \operatorname{set}(\) add-effects-of op)
        \(\Longrightarrow s^{\prime} v=\) Some True
    and \(\bigwedge v\).
        \(v \notin \operatorname{set}(a d d-e f f e c t s-o f o p) \wedge v \in \operatorname{set}(\) delete-effects-of op)
        \(\Longrightarrow s^{\prime} v=\) Some False
    and \(\wedge v\).
        \(v \notin\) set \((a d d-e f f e c t s-o f ~ o p) \wedge v \notin\) set (delete-effects-of op)
        \(\Longrightarrow s^{\prime} v=s v\)
proof (auto)
    show \(\bigwedge v\).
        \(v \in\) set (add-effects-of op)
        \(\Longrightarrow s^{\prime} v=\) Some True
        using assms effect--strips-iii-a
        by fast
next
    show \(\bigwedge v\).
        \(v \notin\) set (add-effects-of op)
        \(\Longrightarrow v \in\) set (delete-effects-of op)
        \(\Longrightarrow s^{\prime} v=\) Some False
        using assms effect--strips-iii-b
        by fast
next
    show \(\wedge v\).
        \(v \notin\) set (add-effects-of op)
    \(\Longrightarrow v \notin\) set (delete-effects-of op)
    \(\Longrightarrow s^{\prime} v=s v\)
```

[^3]```
    using assms effect--strips-iii-c
    by metis
qed
```


### 3.2 Parallel Plan Semantics

definition are-all-operators-applicable s ops
$\equiv$ list-all ( $\lambda$ op. is-operator-applicable-in $s$ op) ops
definition are-operator-effects-consistent $o p_{1} o p_{2} \equiv$ let
$a d d_{1}=a d d$-effects-of op $p_{1}$
; $a d d_{2}=$ add-effects-of op ${ }_{2}$
; del $_{1}=$ delete-effects-of op ${ }_{1}$
; del ${ }_{2}=$ delete-effects-of op ${ }_{2}$
in $\neg$ list-ex $\left(\lambda v\right.$. list-ex $((=) v)$ del $\left._{2}\right)$ add $d_{1} \wedge \neg$ list-ex $\left(\lambda v\right.$. list-ex $\left.((=) v) a d d_{2}\right)$
$d e l_{1}$
definition are-all-operator-effects-consistent ops $\equiv$
list-all ( $\lambda$ op. list-all (are-operator-effects-consistent op) ops) ops
definition execute-parallel-operator
:: 'variable strips-state
$\Rightarrow$ 'variable strips-operator list
$\Rightarrow$ 'variable strips-state
where execute-parallel-operator s ops
$\equiv$ foldl $(++) s$ (map (map-of $\circ$ effect-to-assignments) ops)
The parallel STRIPS execution semantics is defined in similar way as the serial STRIPS execution semantics. However, the applicability test is lifted to parallel operators and we additionally test for operator consistency (which was unecessary in the serial case).

```
fun execute-parallel-plan
    :: 'variable strips-state
        => 'variable strips-parallel-plan
        #'variable strips-state
    where execute-parallel-plan s[]=s
    | execute-parallel-plan s (ops # opss) = (if
                are-all-operators-applicable s ops
                \wedge ~ a r e - a l l - o p e r a t o r - e f f e c t s - c o n s i s t e n t ~ o p s
    then execute-parallel-plan (execute-parallel-operator s ops) opss
    else s)
definition are-operators-interfering op ( op 2
    \equivlist-ex (\lambdav.list-ex ((=)v) (delete-effects-of op p)) (precondition-of op 2)
        \vee \mp@code { l i s t - e x ~ ( ~ } \lambda v . l i s t - e x ~ ( ( = ) ~ v ) ~ ( p r e c o n d i t i o n - o f ~ o p ~ 1 ) ) ~ ( d e l e t e - e f f e c t s - o f ~ o p ~ 2 ) ~
```

primrec are-all-operators-non-interfering
:: 'variable strips-operator list $\Rightarrow$ bool

```
where are-all-operators-non-interfering [] = True
| are-all-operators-non-interfering (op \# ops)
    \(=\left(\right.\) list-all \(\left(\lambda o p^{\prime} . \neg\right.\) are-operators-interfering op op \(\left.{ }^{\prime}\right)\) ops
    \(\wedge\) are-all-operators-non-interfering ops)
```

Since traces mirror the execution semantics, the same is true for the definition of parallel STRIPS plan traces.

```
fun trace-parallel-plan-strips
    \(::\) 'variable strips-state \(\Rightarrow\) 'variable strips-parallel-plan \(\Rightarrow\) 'variable strips-state list
    where trace-parallel-plan-strips \(s[]=[s]\)
    | trace-parallel-plan-strips \(s\) (ops \# opss) \(=s \#\) (if
        are-all-operators-applicable s ops
        \(\wedge\) are-all-operator-effects-consistent ops
    then trace-parallel-plan-strips (execute-parallel-operator sops) opss
    else [])
```

Similarly, the definition of parallel solutions requires that the parallel execution semantics transforms the initial problem into the goal state of the problem and that every operator of every parallel operator in the parallel plan is an operator that is defined in the problem description.

```
definition is-parallel-solution-for-problem
    where is-parallel-solution-for-problem \Pi \pi
    \equiv(strips-problem.goal-of \Pi) \subseteqm execute-parallel-plan
    (strips-problem.initial-of \Pi) \pi
    \wedge list-all (\lambdaops. list-all (\lambdaop.
        ListMem op (strips-problem.operators-of \Pi)) ops) \pi
```

lemma are-all-operators-applicable-set:
are-all-operators-applicable s ops
$\longleftrightarrow(\forall$ op $\in$ set ops. $\forall v \in$ set (precondition-of op). s $v=$ Some True)
unfolding are-all-operators-applicable-def
STRIPS-Representation.is-operator-applicable-in-def list-all-iff
by presburger
lemma are-all-operators-applicable-cons:
assumes are-all-operators-applicable s (op \# ops)
shows is-operator-applicable-in s op
and are-all-operators-applicable s ops
proof -
from assms have a: list-all ( $\lambda o p$. is-operator-applicable-in sop) (op \# ops)
unfolding are-all-operators-applicable-def is-operator-applicable-in-def
STRIPS-Representation.is-operator-applicable-in-def
by blast
then have is-operator-applicable-in $s$ op
by fastforce
moreover \{

```
    from a have list-all (\lambdaop. is-operator-applicable-in s op) ops
        by simp
        then have are-all-operators-applicable s ops
        using are-all-operators-applicable-def is-operator-applicable-in-def
            STRIPS-Representation.is-operator-applicable-in-def
        by blast
    }
    ultimately show is-operator-applicable-in s op
    and are-all-operators-applicable s ops
        by fast+
qed
lemma are-operator-effects-consistent-set:
    assumes op }\in\mathrm{ set ops
    and op 2}\in\mathrm{ set ops
    shows are-operator-effects-consistent op p
        =(set (add-effects-of op p})\cap\mathrm{ set (delete-effects-of op p})={
        set (delete-effects-of op 1) \cap set (add-effects-of op ( ) ={})
    proof -
    have (\neglist-ex ( }\lambdav.list-ex ((=)v) (delete-effects-of op 2)) (add-effects-of op ( ) )
        =(set (add-effects-of op 1 ) \cap set (delete-effects-of op 2 ) = {})
        using list-ex-intersection[of delete-effects-of op 2 add-effects-of op 1]
        by meson
    moreover have (\neglist-ex (\lambdav.list-ex ((=)v) (add-effects-of op 2)) (delete-effects-of
op
        =(set (delete-effects-of op 1 ) \cap set (add-effects-of op }\mp@subsup{\mp@code{L}}{2}{})={}
        using list-ex-intersection[of add-effects-of op 2 delete-effects-of op p
        by meson
    ultimately show are-operator-effects-consistent op (op op
        =(set (add-effects-of op }\mp@subsup{1}{1}{})\cap\mathrm{ set (delete-effects-of op p})={
        set (delete-effects-of op }\mp@subsup{1}{1}{})\cap\mathrm{ set (add-effects-of op 2)}={}
        unfolding are-operator-effects-consistent-def
        by presburger
    qed
lemma are-all-operator-effects-consistent-set:
    are-all-operator-effects-consistent ops
    \longleftrightarrow(\forallo\mp@subsup{p}{1}{}\in\mathrm{ set ops. }\forallo\mp@subsup{p}{2}{}\in\mathrm{ set ops.}
    (set (add-effects-of op 1 ) \cap set (delete-effects-of op 2 ) = {})
        \wedge (set (delete-effects-of op ( ) \cap set (add-effects-of op 2) ={}))
proof -
    {
    fix }o\mp@subsup{p}{1}{}O\mp@subsup{p}{2}{
    assume o\mp@subsup{p}{1}{}\in set ops and op 的\in set ops
    hence are-operator-effects-consistent op ( op 2
        =(set (add-effects-of op }\mp@subsup{1}{1}{})\cap\mathrm{ set (delete-effects-of op p})={
            set (delete-effects-of op ( ) \cap set (add-effects-of op 2) ={})
        using are-operator-effects-consistent-set[of op (ops op %
        by fast
```

```
    }
    thus ?thesis
        unfolding are-all-operator-effects-consistent-def list-all-iff
        by force
    qed
lemma are-all-effects-consistent-tail:
    assumes are-all-operator-effects-consistent (op # ops)
    shows are-all-operator-effects-consistent ops
    proof -
    from assms
    have a: list-all (\lambdaop'. list-all (are-operator-effects-consistent op ')
        (Cons op ops)) (Cons op ops)
        unfolding are-all-operator-effects-consistent-def
        by blast
    then have b-1: list-all (are-operator-effects-consistent op) (op # ops)
    and b-2: list-all (\lambdaop'. list-all (are-operator-effects-consistent op') (op # ops))
ops
    by force+
    then have list-all (are-operator-effects-consistent op) ops
    by simp
    moreover
    {
        fix z
            assume z\in set (Cons op ops)
            and list-all (are-operator-effects-consistent z) (op # ops)
            then have list-all (are-operator-effects-consistent z) ops
                by auto
    }
    then have list-all (\lambdaop'. list-all (are-operator-effects-consistent op') ops) ops
                using list.pred-mono-strong[of
                    (\lambdaop'. list-all (are-operator-effects-consistent op') (op # ops))
                    Cons op ops (\lambdaop'. list-all (are-operator-effects-consistent op') ops)
                ] a
            by fastforce
    }
    ultimately have list-all (are-operator-effects-consistent op) ops
        ^ list-all (\lambdaop'. list-all (are-operator-effects-consistent op') ops) ops
        by blast
    then show ?thesis
        using are-all-operator-effects-consistent-def
        by fast
    qed
lemma are-all-operators-non-interfering-tail:
    assumes are-all-operators-non-interfering (op # ops)
    shows are-all-operators-non-interfering ops
    using assms
```

unfolding are-all-operators-non-interfering-def
by $\operatorname{simp}$
lemma are-operators-interfering-symmetric:
assumes are-operators-interfering op ${ }_{1} o p_{2}$
shows are-operators-interfering op ${ }_{2} o p_{1}$
using assms
unfolding are-operators-interfering-def list-ex-iff
by fast

- A small technical characterizing operator lists with property. We show that pairs of distinct operators which interfere with one another cannot both be contained in the corresponding operator set.
lemma are-all-operators-non-interfering-set-contains-no-distinct-interfering-operator-pairs:
assumes are-all-operators-non-interfering ops
and are-operators-interfering op $p_{1} o p_{2}$
and $o p_{1} \neq o p_{2}$
shows $o p_{1} \notin$ set ops $\vee$ op $p_{2} \notin$ set ops
using assms
proof (induction ops)
case (Cons op ops)
thm Cons.IH[OF - Cons.prems(2, 3)]
have $n b_{1}: \forall o p^{\prime} \in$ set ops. $\neg$ are-operators-interfering op op ${ }^{\prime}$
and $n b_{2}$ : are-all-operators-non-interfering ops
using Cons.prems(1)
unfolding are-all-operators-non-interfering.simps(2) list-all-iff
by blast+
then consider $(A) o p=o p_{1}$
(B) op $=o p_{2}$
| $(C) o p \neq o p_{1} \wedge o p \neq o p_{2}$
by blast
thus ?case
proof (cases)
case $A$
\{
assume $o p_{2} \in \operatorname{set}(o p \# o p s)$
then have $o p_{2} \in$ set ops
using Cons.prems(3) A
by force
then have $\neg$ are-operators-interfering $o p_{1} o p_{2}$
using $n b_{1} A$
by fastforce
hence False
using Cons.prems(2)..
\}
thus ?thesis
by blast
next
case $B$

```
        {
            assume op 
            then have op
            using Cons.prems(3) B
            by force
            then have }\neg\mathrm{ are-operators-interfering op 
            using nb l B are-operators-interfering-symmetric
            by blast
        hence False
            using Cons.prems(2)..
        }
        thus ?thesis
        by blast
        next
        case C
        thus ?thesis
        using Cons.IH[OF nb 2 Cons.prems(2, 3)]
        by force
    qed
qed simp
lemma execute-parallel-plan-precondition-cons-i:
fixes s:: ('variable, bool) state
    assumes \negare-operators-interfering op op'
        and is-operator-applicable-in s op
    and is-operator-applicable-in s op'
shows is-operator-applicable-in (s++ map-of (effect-to-assignments op))op'
proof -
    let ? s}\mp@subsup{s}{}{\prime}=s++\mathrm{ map-of (effect-to-assignments op)
    - TODO slightly hackish to exploit the definition of execute-operator, but
we otherwise have to rewrite theorem operator-effect--strips (which is a todo as of
now).
    {
        have a:? }\mp@subsup{s}{}{\prime}=s>>o
            by (simp add: execute-operator-def)
            then have }\v.v\inset (add-effects-of op)\Longrightarrow?s'v=Some True
            and \v.v\not\inset (add-effects-of op) ^v\inset (delete-effects-of op)\Longrightarrow?s'
v=Some False
            and }\Lambdav.v\not\in\mathrm{ set (add-effects-of op) }\wedgev\not\in\mathrm{ set (delete-effects-of op) }\Longrightarrow?s
v=sv
            using operator-effect--strips
            by metis+
    }
    note a= this
    - TODO refactor lemma not-have-interference-set.
    {
    fix v
    assume \alpha:v\in set (precondition-of op')
```

```
    {
        fix v
        have }\neg\mathrm{ list-ex ((=)v) (delete-effects-of op)
        = list-all ( }\lambda\mp@subsup{v}{}{\prime}.\negv=\mp@subsup{v}{}{\prime})(\mathrm{ delete-effects-of op)
        using not-list-ex-equals-list-all-not[
            where P=(=)v and xs=delete-effects-of op]
        by blast
    } moreover {
        from assms(1)
    have \neglist-ex (\lambdav.list-ex ((=) v) (delete-effects-of op)) (precondition-of op')
        unfolding are-operators-interfering-def
        by blast
    then have list-all ( }\lambdav.\neglist-ex ((=)v) (delete-effects-of op)) (precondition-of
op')
        using not-list-ex-equals-list-all-not[
            where P=\lambdav.list-ex ((=)v) (delete-effects-of op) and xs=precondition-of
op]
        by blast
    }
    ultimately have }\beta\mathrm{ :
    list-all ( }\lambdav.list-all (\lambda\mp@subsup{v}{}{\prime}.\negv=\mp@subsup{v}{}{\prime})(\mathrm{ delete-effects-of op)) (precondition-of op')
        by presburger
    moreover {
    fix v
    have list-all ( }\lambda\mp@subsup{v}{}{\prime}.\negv=\mp@subsup{v}{}{\prime})(\mathrm{ delete-effects-of op)
        =(\forall\mp@subsup{v}{}{\prime}\in\operatorname{set}(\mathrm{ delete-effects-of op).}\negv=\mp@subsup{v}{}{\prime})
        using list-all-iff [where P=\lambdav'.\negv= v' and x=delete-effects-of op]
    }
    ultimately have }\forallv\in\operatorname{set}\mathrm{ (precondition-of op'). }\forall\mp@subsup{v}{}{\prime}\in\mathrm{ set (delete-effects-of
op).}\negv=\mp@subsup{v}{}{\prime
    using \beta list-all-iff[
        where P}=\lambdav.list-all ( \lambda\mp@subsup{v}{}{\prime}.\negv=\mp@subsup{v}{}{\prime})(\mathrm{ delete-effects-of op)
            and x=precondition-of op]
        by presburger
    then have v}\not\in\mathrm{ set (delete-effects-of op)
        using }
        by fast
}
note b = this
{
    fix v
    assume a:v set (precondition-of op')
    have list-all (\lambdav. s v = Some True) (precondition-of op')
        using assms(3)
        unfolding is-operator-applicable-in-def
        STRIPS-Representation.is-operator-applicable-in-def
        by presburger
    then have }\forallv\in\mathrm{ set (precondition-of op'). s v}=\mathrm{ Some True
```

using list-all-iff[where $P=\lambda v$. s $v=$ Some True and $x=$ precondition-of $o p$ ]
by blast
then have $s v=$ Some True
using $a$
by blast
\}
note $c=$ this
\{
fix $v$
assume $d: v \in$ set (precondition-of op ${ }^{\prime}$ )
then have ? $s^{\prime} v=$ Some True
proof (cases $v \in$ set (add-effects-of op))
case True
then show ?thesis
using $a$
by blast
next
case $e$ : False
then show ?thesis
proof (cases $v \in$ set (delete-effects-of op))
case True
then show ?thesis
using $\operatorname{assms}(1) b d$
by fast
next
case False
then have ? $s^{\prime} v=s v$
using $a e$
by blast
then show ?thesis
using $c d$
by presburger
qed
qed
\}
then have list-all ( $\lambda v$. ? $s^{\prime} v=$ Some True) (precondition-of op')
using list-all-iff[where $P=\lambda v$. ? $s^{\prime} v=$ Some True and $x=$ precondition-of $o p]$
by blast
then show ?thesis
unfolding is-operator-applicable-in-def
STRIPS-Representation.is-operator-applicable-in-def
by auto
qed

- The third assumption are-all-operators-non-interfering ( $a \neq$ ops $)^{\prime \prime}$ is not part of the precondition of but is required for the proof of the subgoal hat applicable is maintained.

```
lemma execute-parallel-plan-precondition-cons:
    fixes a :: 'variable strips-operator
    assumes are-all-operators-applicable s (a # ops)
        and are-all-operator-effects-consistent (a # ops)
        and are-all-operators-non-interfering (a # ops)
    shows are-all-operators-applicable (s ++ map-of (effect-to-assignments a)) ops
        and are-all-operator-effects-consistent ops
        and are-all-operators-non-interfering ops
    using are-all-effects-consistent-tail[OF assms(2)]
        are-all-operators-non-interfering-tail[OF assms(3)]
    proof -
    let ?s' = s++ map-of (effect-to-assignments a)
    have n\mp@subsup{b}{1}{}:\forallop\in set (a# ops). is-operator-applicable-in s op
        using assms(1) are-all-operators-applicable-set
        unfolding are-all-operators-applicable-def is-operator-applicable-in-def
            STRIPS-Representation.is-operator-applicable-in-def list-all-iff
        by blast
    have n\mp@subsup{b}{2}{}:\forallop\in set ops.\negare-operators-interfering a op
            using assms(3)
        unfolding are-all-operators-non-interfering-def list-all-iff
        by simp
    have nb}\mp@subsup{3}{3}{}\mathrm{ : is-operator-applicable-in s a
            using assms(1) are-all-operators-applicable-set
            unfolding are-all-operators-applicable-def is-operator-applicable-in-def
                STRIPS-Representation.is-operator-applicable-in-def list-all-iff
            by force
    {
    fix op
    assume op-in-ops:op \in set ops
    hence is-operator-applicable-in ?s' op
                using execute-parallel-plan-precondition-cons-i[of a op] nb ll}n\mp@subsup{b}{2}{}n\mp@subsup{b}{3}{
            by force
    }
    then show are-all-operators-applicable ?s' ops
        unfolding are-all-operators-applicable-def list-all-iff
            is-operator-applicable-in-def
        by blast
qed
lemma execute-parallel-operator-cons[simp]:
        execute-parallel-operator s (op # ops)
        = execute-parallel-operator (s ++ map-of (effect-to-assignments op)) ops
    unfolding execute-parallel-operator-def
    by simp
lemma execute-parallel-operator-cons-equals:
assumes are-all-operators-applicable s ( \(a\) \# ops)
        and are-all-operator-effects-consistent (a # ops)
    and are-all-operators-non-interfering (a # ops)
```

```
shows execute-parallel-operator s (a # ops)
    = execute-parallel-operator (s++ map-of (effect-to-assignments a)) ops
proof -
    let ? s' = s++ map-of (effect-to-assignments a)
    {
        from assms(1, 2)
        have execute-parallel-operator s (Cons a ops)
            = foldl (++)s(map (map-of ○ effect-to-assignments) (Cons a ops))
            unfolding execute-parallel-operator-def
            by presburger
        also have ... = foldl (++) (?s')
            (map (map-of o effect-to-assignments) ops)
            by auto
        finally have execute-parallel-operator s(Cons a ops)
            = foldl (++) (?s')
                (map (map-of o effect-to-assignments) ops)
            using execute-parallel-operator-def
            by blast
    }
```

    - NOTE the precondition of for \(a \#\) ops is also true for the tail list and state
    $? s^{\prime}$ as shown in lemma. Hence the precondition for the r.h.s. of the goal also holds.
moreover have execute-parallel-operator ?s' ops
$=$ foldl $(++)(s++($ map-of $\circ$ effect-to-assignments $) a)$
(map (map-of o effect-to-assignments) ops)
by (simp add: execute-parallel-operator-def)
ultimately show ?thesis
by force
qed

- We show here that following the lemma above, executing one operator of a parallel operator can be replaced by a (single) STRIPS operator execution.
corollary execute-parallel-operator-cons-equals-corollary:
assumes are-all-operators-applicable s ( $a \mathrm{\#}$ ops)
shows execute-parallel-operator s ( $a$ \# ops)
$=$ execute-parallel-operator $(s \gg a)$ ops
proof -
let ? $s^{\prime}=s++$ map-of (effect-to-assignments a)
from assms
have execute-parallel-operator $s$ ( $a \#$ ops)
$=$ execute-parallel-operator $(s++$ map-of (effect-to-assignments a)) ops using execute-parallel-operator-cons-equals by $\operatorname{simp}$
moreover have ? $s^{\prime}=s \gg a$
unfolding execute-operator-def
by $\operatorname{simp}$
ultimately show?thesis
by argo
qed

```
lemma effect-to-assignments-simp[simp]: effect-to-assignments op
    =map (\lambdav. (v,True)) (add-effects-of op)@ map (\lambdav. (v,False)) (delete-effects-of
op)
    by (simp add: effect-to-assignments-i)
lemma effect-to-assignments-set-is[simp]:
    set (effect-to-assignments op) ={((v,a),True)|va.(v,a)\in set (add-effects-of
op) }
    \cup \{ ( ( v , a ) , \text { False )\|va.(v,a) G set (delete-effects-of op) \}}
proof -
    obtain as where effect--strips op = as
            and as = map (\lambdav. (v, True))(add-effects-of op)
                @ map (\lambdav. (v, False)) (delete-effects-of op)
            unfolding effect--strips-def
            by blast
    moreover have as
    =map (\lambdav. (v, True)) (add-effects-of op)@ map (\lambdav. (v, False)) (delete-effects-of
op)
            using calculation(2)
            unfolding map-append map-map comp-apply
            by auto
    moreover have effect-to-assignments op = as
            unfolding effect-to-assignments-def calculation(1, 2)
            by auto
    ultimately show ?thesis
            unfolding set-map
            by auto
qed
corollary effect-to-assignments-construction-from-function-graph:
    assumes set (add-effects-of op) \cap set (delete-effects-of op) = {}
    shows effect-to-assignments op = map
        (\lambdav. (v, if ListMem v (add-effects-of op) then True else False))
        (add-effects-of op @ delete-effects-of op)
    and effect-to-assignments op = map
        ( \lambdav. (v, if ListMem v (delete-effects-of op) then False else True))
        (add-effects-of op @ delete-effects-of op)
    proof -
    let ?f = \lambdav.(v, if ListMem v (add-effects-of op) then True else False)
            and ?g = \lambdav.(v, if ListMem v (delete-effects-of op) then False else True)
    {
            have map ?f (add-effects-of op @ delete-effects-of op)
            = map ?f (add-effects-of op)@ map ?f (delete-effects-of op)
            using map-append
            by fast
            - TODO slow.
            hence effect-to-assignments op = map ?f (add-effects-of op @ delete-effects-of
op)
```

```
            using ListMem-iff assms
            by fastforce
    } moreover {
        have map ?g (add-effects-of op @ delete-effects-of op)
            = map ?g (add-effects-of op) @ map ?g (delete-effects-of op)
            using map-append
            by fast
            TODO slow.
    hence effect-to-assignments op = map ?g (add-effects-of op @ delete-effects-of
op)
            using ListMem-iff assms
            by fastforce
    }
    ultimately show effect-to-assignments op = map
    ( \lambdav. (v, if ListMem v (add-effects-of op) then True else False))
    (add-effects-of op @ delete-effects-of op)
    and effect-to-assignments op = map
    (\lambdav. (v, if ListMem v (delete-effects-of op) then False else True))
    (add-effects-of op @ delete-effects-of op)
    by blast+
qed
corollary map-of-effect-to-assignments-is-none-if:
    assumes }\negv\in\mathrm{ set (add-effects-of op)
    and }\negv\in\mathrm{ set (delete-effects-of op)
shows map-of (effect-to-assignments op) v}=\mathrm{ None
proof -
    let ?l = effect-to-assignments op
    {
        have set ?l ={(v, True) |v.v\in set (add-effects-of op) }
            \cup \{ ( v , \text { False) \| v.v 部 (delete-effects-of op)\}}
            by auto
    then have fst ' set ?l
            =(fst'{(v,True)|v.v\in set (add-effects-of op)})
                \cup(fst'{(v, False)|v.v\in set (delete-effects-of op)})
            using image-Un[of fst {(v, True)|v.v\in set (add-effects-of op)}
                {(v, False)|v.v\in set (delete-effects-of op)}]
            by presburger
            - TODO slow.
    also have ... = (fst'( }\lambdav.(v,\mathrm{ True))' set (add-effects-of op))
            \cup(fst'(\lambdav. (v, False))' set (delete-effects-of op))
            using setcompr-eq-image[of \lambdav. (v, True) \lambdav.v\in set (add-effects-of op)]
                setcompr-eq-image[of \lambdav.(v, False) \lambdav.v \in set (delete-effects-of op)]
            by simp
                            - TODO slow.
                            also have ... = id'set (add-effects-of op) \cupid'set (delete-effects-of op)
                by force
            - TODO slow.
            finally have fst'set ?l = set (add-effects-of op) \cup set (delete-effects-of op)
```

```
        by auto
    hence v\not\infst ' set ?l
        using assms(1, 2)
        by blast
    }
    thus ?thesis
        using map-of-eq-None-iff[of ?l v]
        by blast
qed
lemma execute-parallel-operator-positive-effect-if-i:
    assumes are-all-operators-applicable s ops
        and are-all-operator-effects-consistent ops
    and op\in set ops
    and}v\in\operatorname{set (add-effects-of op)
shows map-of (effect-to-assignments op) v=Some True
proof -
    let ?f = \lambdax. if ListMem x (add-effects-of op) then True else False
    and ?l'= map (\lambdav.(v, if ListMem v (add-effects-of op) then True else False))
        (add-effects-of op @ delete-effects-of op)
    have set (add-effects-of op) }={
        using assms(4)
        by fastforce
    moreover {
        have set (add-effects-of op) \cap set (delete-effects-of op) = {}
        using are-all-operator-effects-consistent-set assms(2, 3)
        by fast
    moreover have effect-to-assignments op = ?l'
        using effect-to-assignments-construction-from-function-graph(1) calculation
        by fast
    ultimately have map-of (effect-to-assignments op) = map-of ?l'
        by argo
    }
    ultimately have map-of (effect-to-assignments op) v=Some (?f v)
    using Map-Supplement.map-of-from-function-graph-is-some-if[
        of - ? ?f,OF - assms(4)]
    by simp
    thus ?thesis
        using ListMem-iff assms(4)
        by metis
    qed
lemma execute-parallel-operator-positive-effect-if:
    fixes ops
    assumes are-all-operators-applicable s ops
    and are-all-operator-effects-consistent ops
    and op\in set ops
    and}v\in\operatorname{set (add-effects-of op)
shows execute-parallel-operator s ops v = Some True
```

```
proof -
    let ?l = map (map-of o effect-to-assignments) ops
    have set-l-is: set ?l = (map-of \circ effect-to-assignments)' set ops
        using set-map
        by fastforce
    {
        let ?m = (map-of ○ effect-to-assignments) op
        have ?m}\in\mathrm{ set ?l
            using assms(3) set-l-is
            by blast
    moreover have ?m v = Some True
            using execute-parallel-operator-positive-effect-if-i[OF assms]
            by fastforce
    ultimately have }\existsm\in\mathrm{ set ?l. m v=Some True
            by blast
}
moreover {
    fix m'
    assume m'\in set ?l
    then obtain op'
    where op'-in-set-ops:op' }\in\mathrm{ set ops
                            and m'-is: m' = (map-of ○ effect-to-assignments) op'
    by auto
    then have set (add-effects-of op) \cap set (delete-effects-of op')={}
    using assms(2, 3) are-all-operator-effects-consistent-set[of ops]
    by blast
then have v\not\in set (delete-effects-of op')
    using assms(4)
    by blast
then consider (v-in-set-add-effects) v\in set (add-effects-of op')
            | (otherwise) }\negv\in\operatorname{set}(add-effects-of op') ^ \negv\in set (delete-effects-of op'
            by blast
hence m'v = Some True \vee m'v=None
    proof (cases)
                case v-in-set-add-effects
            - TODO slow.
                thus ?thesis
                    using execute-parallel-operator-positive-effect-if-i[
                    OF assms(1, 2) op'-in-set-ops, of v] m'-is
                by simp
next
                case otherwise
                then have }\negv\in\operatorname{set (add-effects-of op')
                    and }\negv\in\mathrm{ set (delete-effects-of op')
                    by blast+
                thus ?thesis
                    using map-of-effect-to-assignments-is-none-if[of vopl] m'-is
                by fastforce
            qed
```

```
    }
    - TODO slow.
    ultimately show ?thesis
        unfolding execute-parallel-operator-def
        using foldl-map-append-is-some-if[of s v True ?l]
        by meson
qed
lemma execute-parallel-operator-negative-effect-if-i:
    assumes are-all-operators-applicable s ops
    and are-all-operator-effects-consistent ops
    and op \in set ops
    and}v\in\mathrm{ set (delete-effects-of op)
shows map-of (effect-to-assignments op) v= Some False
proof -
    let ?f = \lambdax. if ListMem x (delete-effects-of op) then False else True
            and ?l'= map (\lambdav. (v, if ListMem v (delete-effects-of op) then False else
True))
            (add-effects-of op @ delete-effects-of op)
    have set (delete-effects-of op @ add-effects-of op) }={
        using assms(4)
        by fastforce
    moreover have v\in set (delete-effects-of op @ add-effects-of op)
        using assms(4)
        by simp
    moreover {
    have set (add-effects-of op) \cap set (delete-effects-of op)={}
        using are-all-operator-effects-consistent-set assms(2, 3)
        by fast
    moreover have effect-to-assignments op =? ?'
            using effect-to-assignments-construction-from-function-graph(2) calculation
            by blast
            ultimately have map-of (effect-to-assignments op) = map-of ?l'
            by argo
    }
    ultimately have map-of (effect-to-assignments op) v=Some (?f v)
        using Map-Supplement.map-of-from-function-graph-is-some-if[
            of add-effects-of op @ delete-effects-of op v ?f]
    by force
    thus ?thesis
        using assms(4)
        unfolding ListMem-iff
        by presburger
qed
lemma execute-parallel-operator-negative-effect-if:
assumes are-all-operators-applicable s ops and are-all-operator-effects-consistent ops and \(o p \in\) set ops
```

```
    and}v\in\mathrm{ set (delete-effects-of op)
shows execute-parallel-operator s ops v}=\mathrm{ Some False
proof -
    let ?l = map (map-of ○ effect-to-assignments) ops
    have set-l-is: set ?l = (map-of ○ effect-to-assignments)' set ops
        using set-map
    by fastforce
{
    let ?m}=(\mathrm{ map-of ○ effect-to-assignments) op
    have ?m}\in set ?l 
        using assms(3) set-l-is
        by blast
    moreover have ?m v = Some False
        using execute-parallel-operator-negative-effect-if-i[OF assms]
        by fastforce
    ultimately have }\existsm\in\mathrm{ set ?l. m v = Some False
        by blast
}
moreover {
    fix m'
    assume m' }\in\mathrm{ set?l
    then obtain op'
        where o\mp@subsup{p}{}{\prime}-in-set-ops:op'}\in\mathrm{ set ops
            and m'-is: m' = (map-of \circ effect-to-assignments)op'
        by auto
    then have set (delete-effects-of op) \cap set (add-effects-of op')={}
        using assms(2, 3) are-all-operator-effects-consistent-set[of ops]
        by blast
    then have v\not\in set (add-effects-of op')
        using assms(4)
        by blast
    then consider (v-in-set-delete-effects) v\in set (delete-effects-of op')
        | (otherwise) }\negv\in\operatorname{set (add-effects-of op')}\wedge\negv\in set (delete-effects-of op'
        by blast
    hence m'v=Some False \vee m'v=None
    proof (cases)
        case v-in-set-delete-effects
            - TODO slow.
            thus ?thesis
                using execute-parallel-operator-negative-effect-if-i[
                    OF assms(1, 2) op'-in-set-ops, of v] m'-is
        by simp
    next
        case otherwise
        then have }\negv\in\operatorname{set (add-effects-of op')
            and }\negv\in\mathrm{ set (delete-effects-of op')
            by blast+
            thus ?thesis
                using map-of-effect-to-assignments-is-none-if[of vop] m'-is
```

```
                by fastforce
            qed
    }
    - TODO slow.
    ultimately show ?thesis
        unfolding execute-parallel-operator-def
        using foldl-map-append-is-some-if[of s v False ?l]
        by meson
qed
lemma execute-parallel-operator-no-effect-if:
assumes }\forallop\in\mathrm{ set ops. }\negv\in\mathrm{ set (add-effects-of op) }\wedge\negv\in\mathrm{ set (delete-effects-of
op)
shows execute-parallel-operator s ops v = sv
using assms
unfolding execute-parallel-operator-def
proof (induction ops arbitrary: s)
    case (Cons a ops)
    let ?f = map-of ○ effect-to-assignments
    {
    have v\not\in set (add-effects-of a) ^v vet (delete-effects-of a)
        using Cons.prems(1)
        by force
    then have ?f a v=None
            using map-of-effect-to-assignments-is-none-if[of v a]
            by fastforce
    then have v}\not\in\operatorname{dom}(?f a
            by blast
    hence (s++?fa) v=sv
            using map-add-dom-app-simps(3)[of v ?f a s]
            by blast
    }
    moreover {
    have }\forallop\inset ops.v\not\in set (add-effects-of op) ^v\not\in set (delete-effects-of op
            using Cons.prems(1)
            by simp
    hence foldl (++) (s++ ?f a) (map ?f ops) v=(s++ ?f a) v
            using Cons.IH[of s ++ ?f a]
            by blast
}
moreover {
    have map ?f (a # ops) = ?f a # map ?f ops
        by force
    then have foldl (++)s(map ?f (a # ops))
        = foldl (++) (s++ ?f a)(map ?f ops)
        using foldl-Cons
        by force
    }
    ultimately show ?case
```

```
    by argo
qed fastforce
corollary execute-parallel-operators-strips-none-if:
    assumes }\forallop\in\mathrm{ set ops. }\negv\in\mathrm{ set (add-effects-of op) }\wedge\negv\in\mathrm{ set (delete-effects-of
op)
    and sv=None
    shows execute-parallel-operator s ops v = None
    using execute-parallel-operator-no-effect-if[OF assms(1)] assms(2)
    by simp
corollary execute-parallel-operators-strips-none-if-contraposition:
    assumes \negexecute-parallel-operator s ops v=None
    shows (\existsop\in set ops.v\in set (add-effects-of op) \veev\in set (delete-effects-of op))
    v v}\not=Non
    proof -
    let ?P = (\forall op\in set ops. }\negv\in\operatorname{set}(\mathrm{ add-effects-of op) }\wedge\negv\in set (delete-effects-of
op))
        \wedge v = None
        and ?Q = execute-parallel-operator s ops v = None
    have ?P\Longrightarrow??
        using execute-parallel-operators-strips-none-if[of ops v s]
        by blast
    then have \neg?P
        using contrapos-nn[of ?Q ?P]
        using assms
        by argo
    thus ?thesis
        by meson
    qed
```

We will now move on to showing the equivalent to theorem in. Under the condition that for a list of operators ops all operators in the corresponding set are applicable in a given state $s$ and all operator effects are consistent, if an operator $o p$ exists with $o p \in$ set ops and with $v$ being an add effect of $o p$, then the successor state

$$
s^{\prime} \equiv \text { execute-parallel-operator } s \text { ops }
$$

will evaluate $v$ to true, that is

$$
\text { execute-parallel-operator } s \text { ops } v=\text { Some True }
$$

Symmetrically, if $v$ is a delete effect, we have

```
execute-parallel-operator s ops v}=\mathrm{ Some False
```

under the same condition as for the positive effect. Lastly, if $v$ is neither an add effect nor a delete effect for any operator in the operator set corresponding to ops, then the state after parallel operator execution remains unchanged, i.e.

```
execute-parallel-operator s ops v}=s
```

theorem execute-parallel-operator-effect:
assumes are-all-operators-applicable s ops
and are-all-operator-effects-consistent ops
shows op set ops $\wedge v \in$ set (add-effects-of op)
$\longrightarrow$ execute-parallel-operator s ops $v=$ Some True
and $o p \in$ set ops $\wedge v \in$ set (delete-effects-of op)
$\longrightarrow$ execute-parallel-operator $s$ ops $v=$ Some False
and $(\forall o p \in$ set ops.
$v \notin \operatorname{set}($ add-effects-of op) $\wedge v \notin$ set (delete-effects-of op))
$\longrightarrow$ execute-parallel-operator $s$ ops $v=s v$
using execute-parallel-operator-positive-effect-if[OF assms]
execute-parallel-operator-negative-effect-if[OF assms]
execute-parallel-operator-no-effect-if[of ops $v s$ ]
by blast+
lemma is-parallel-solution-for-problem-operator-set:
fixes $\Pi$ :: 'a strips-problem
assumes is-parallel-solution-for-problem $\Pi \pi$
and ops $\in$ set $\pi$
and $o p \in$ set ops
shows op $\in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)$
proof -
have $\forall$ ops $\in$ set $\pi . \forall o p \in$ set ops. op $\in$ set (strips-problem.operators-of $\Pi$ )
using assms(1)
unfolding is-parallel-solution-for-problem-def list-all-iff ListMem-iff..
thus ?thesis
using $\operatorname{assms}(2,3)$
by fastforce
qed
lemma trace-parallel-plan-strips-not-nil: trace-parallel-plan-strips $I \pi \neq[]$
proof (cases $\pi$ )
case (Cons a list)
then show?thesis
by (cases are-all-operators-applicable $I(h d \pi) \wedge$ are-all-operator-effects-consistent (hd $\pi$ )
, $\operatorname{simp} p+$
qed $\operatorname{simp}$
corollary length-trace-parallel-plan-gt- $0[$ simp $]: 0<$ length (trace-parallel-plan-strips $I \pi$ )
using trace-parallel-plan-strips-not-nil..
corollary length-trace-minus-one-lt-length-trace[simp]:
length (trace-parallel-plan-strips $I \pi$ ) $-1<$ length (trace-parallel-plan-strips $I$
$\pi$ )
using diff-less[OF - length-trace-parallel-plan-gt-0]
by auto
lemma trace-parallel-plan-strips-head-is-initial-state:
trace-parallel-plan-strips $I \pi!0=I$
proof (cases $\pi$ )
case (Cons a list)
then show ?thesis
by (cases are-all-operators-applicable I a $\wedge$ are-all-operator-effects-consistent $a, \operatorname{simp}+$ )
qed $\operatorname{simp}$
lemma trace-parallel-plan-strips-length-gt-one-if:
assumes $k<$ length (trace-parallel-plan-strips $I \pi$ ) -1
shows $1<$ length (trace-parallel-plan-strips $I \pi$ )
using assms
by linarith

- This lemma simply shows that the last element of a trace-parallel-plan-strips execution step $s$ \# trace-parallel-plan-strips $s^{\prime} \pi$ always is the last element of trace-parallel-plan-strips $s^{\prime} \pi$ since trace-parallel-plan-strips always returns at least a singleton list (even if $\pi=[]$ ).
lemma trace-parallel-plan-strips-last-cons-then: last ( $s \#$ trace-parallel-plan-strips $s^{\prime} \pi$ ) $=$ last (trace-parallel-plan-strips $s^{\prime} \pi$ ) by (cases $\pi$, simp, force)

Parallel plan traces have some important properties that we want to confirm before proceeding. Let $\tau \equiv$ trace-parallel-plan-strips $I \pi$ be a trace for a parallel plan $\pi$ with initial state $I$.
First, all parallel operators ops $=\pi!k$ for any index $k$ with $k<$ length $\tau-1$ (meaning that $k$ is not the index of the last element). must be applicable and their effects must be consistent. Otherwise, the trace would have terminated and ops would have been the last element. This would violate the assumption that $k<$ length $\tau-1$ is not the last index since the index of the last element is length $\tau-1 .{ }^{4}$
lemma trace-parallel-plan-strips-operator-preconditions:
assumes $k<$ length (trace-parallel-plan-strips $I \pi$ ) - 1
shows are-all-operators-applicable (trace-parallel-plan-strips $I \pi!k)(\pi!k)$
$\wedge$ are-all-operator-effects-consistent $(\pi!k)$
using assms

[^4]```
    proof (induction \pi arbitrary: I k)
```

- NOTE Base case yields contradiction with assumption and can be left to automation.
case (Cons a $\pi$ )
then show ?case
proof (cases are-all-operators-applicable I a $\wedge$ are-all-operator-effects-consistent a)


## case True

have trace-parallel-plan-strips-cons: trace-parallel-plan-strips $I(a \neq \pi)$
$=I \#$ trace-parallel-plan-strips (execute-parallel-operator I a) $\pi$
using True
by $\operatorname{simp}$
then show ?thesis
proof (cases $k$ )
case 0
have trace-parallel-plan-strips $I(a \# \pi)!0=I$
using trace-parallel-plan-strips-cons
by $\operatorname{simp}$
moreover have $(a \# \pi)!0=a$
by $\operatorname{simp}$
ultimately show ?thesis
using True 0
by presburger

## next

case (Suc $k^{\prime}$ )
let $? I^{\prime}=$ execute-parallel-operator $I$ a
have trace-parallel-plan-strips $I(a \# \pi)!$ Suc $k^{\prime}=$ trace-parallel-plan-strips ? $I^{\prime} \pi!k^{\prime}$
using trace-parallel-plan-strips-cons
by $\operatorname{simp}$
moreover have $(a \# \pi)!$ Suc $k^{\prime}=\pi!k^{\prime}$
by simp
moreover \{
have length (trace-parallel-plan-strips $I(a \# \pi))$
$=1+$ length (trace-parallel-plan-strips ? $\left.I^{\prime} \pi\right)$
unfolding trace-parallel-plan-strips-cons
by simp
then have $k^{\prime}<$ length (trace-parallel-plan-strips ? $\left.I^{\prime} \pi\right)-1$
using Suc Cons.prems
by fastforce
hence are-all-operators-applicable (trace-parallel-plan-strips ? $I^{\prime} \pi!k^{\prime}$ )
$\wedge$ are-all-operator-effects-consistent ( $\pi!k^{\prime}$ )
using Cons.IH[of $k$ ]
by blast
\}
ultimately show ?thesis
using Suc
by argo

## qed

next
case False
then have trace-parallel-plan-strips $I(a \# \pi)=[I]$
by force
then have length (trace-parallel-plan-strips $I(a \# \pi))-1=0$ by $\operatorname{simp}$

- NOTE Thesis follows from contradiction with assumption.
then show ?thesis
using Cons.prems
by force
qed
qed auto
Another interesting property that we verify below is that elements of the trace store the result of plan prefix execution. This means that for an index $k$ with
$k<$ length (trace-parallel-plan-strips $I \pi$ ), the $k$-th element of the trace is state reached by executing the plan prefix take $k \pi$ consisting of the first $k$ parallel operators of $\pi$.
lemma trace-parallel-plan-plan-prefix:
assumes $k<$ length (trace-parallel-plan-strips $I \pi$ )
shows trace-parallel-plan-strips $I \pi!k=$ execute-parallel-plan $I($ take $k \pi)$
using assms
proof (induction $\pi$ arbitrary: I k)
case (Cons a $\pi$ )
then show?case
proof (cases are-all-operators-applicable I $a \wedge$ are-all-operator-effects-consistent
a)
case True
let $? \sigma=$ trace-parallel-plan-strips $I(a \# \pi)$
and $? I^{\prime}=$ execute-parallel-operator I a
have $\sigma$-equals: ? $\sigma=I \#$ trace-parallel-plan-strips $? I^{\prime} \pi$
using True
by auto
then show ? thesis
proof (cases $k=0$ )
case False
obtain $k^{\prime}$ where $k$-is-suc-of- $k^{\prime}: k=$ Suc $k^{\prime}$
using not0-implies-Suc[OF False]
by blast
then have execute-parallel-plan I (take $k(a \# \pi))$
$=$ execute-parallel-plan ? $I^{\prime}\left(\right.$ take $\left.k^{\prime} \pi\right)$
using True
by simp
moreover have trace-parallel-plan-strips $I(a \# \pi)!k$
$=$ trace-parallel-plan-strips ? $I^{\prime} \pi!k^{\prime}$
using $\sigma$-equals $k$-is-suc-of- $k^{\prime}$
by $\operatorname{simp}$

```
            moreover {
            have }\mp@subsup{k}{}{\prime}<length (trace-parallel-plan-strips (execute-parallel-operator I
a) \pi)
                using Cons.prems \sigma-equals k-is-suc-of-k'
                    by force
                    hence trace-parallel-plan-strips ?I' }\pi!\mp@subsup{k}{}{\prime
                            = execute-parallel-plan ? ?' (take k'}\pi
                    using Cons.IH[of k' ?I']
                        by blast
                }
                ultimately show ?thesis
                    by presburger
            qed simp
        next
            case operator-precondition-violated: False
            then show ?thesis
            proof (cases k=0)
                    case False
                    then have trace-parallel-plan-strips I (a#\pi)=[I]
                        using operator-precondition-violated
                by force
                    moreover have execute-parallel-plan I (take k (a#\pi)) =I
                    using Cons.prems operator-precondition-violated
                    by force
                    ultimately show ?thesis
                    using Cons.prems nth-Cons-0
                    by auto
            qed simp
        qed
    qed simp
lemma length-trace-parallel-plan-strips-lte-length-plan-plus-one:
    shows length (trace-parallel-plan-strips I \pi) \leqlength }\pi+
    proof (induction \pi arbitrary:I)
    case (Cons a \pi)
    then show ?case
    proof (cases are-all-operators-applicable I a ^ are-all-operator-effects-consistent
a)
            case True
            let ?I' = execute-parallel-operator I a
            {
            have trace-parallel-plan-strips I (a # \pi) = I # trace-parallel-plan-strips
?I' }
            using True
            by auto
            then have length (trace-parallel-plan-strips I (a# \pi))
                    = length (trace-parallel-plan-strips ? 'I' }\pi\mathrm{ ) + 1
                    by simp
```

```
            moreover have length (trace-parallel-plan-strips ?I' }\pi\mathrm{ ) }\leq\mathrm{ length }\pi+
                    using Cons.IH[of ?I']
                    by blast
                            ultimately have length (trace-parallel-plan-strips I (a # \pi)) \leqlength (a
# \pi)}+
            by simp
    }
    thus ?thesis
        by blast
    qed auto
qed simp
- Show that \(\pi\) is at least a singleton list.
lemma plan-is-at-least-singleton-plan-if-trace-has-at-least-two-elements:
    assumes k< length (trace-parallel-plan-strips I \pi) - 1
obtains ops \pi' where }\pi=\mathrm{ ops # }\mp@subsup{\pi}{}{\prime
proof -
    let ? }\tau=\mathrm{ trace-parallel-plan-strips I }
    have length ?\tau }\leq\mathrm{ length }\pi+
        using length-trace-parallel-plan-strips-lte-length-plan-plus-one
        by fast
    then have 0< length \pi
        using trace-parallel-plan-strips-length-gt-one-if assms
        by force
    then obtain k' where length }\pi=\mathrm{ Suc k'
        using gr0-implies-Suc
        by meson
    thus ?thesis using that
        using length-Suc-conv[of \pik]
        by blast
qed
- Show that if a parallel plan trace does not have maximum length, in the last state reached through operator execution the parallel operator execution condition was violated.
corollary length-trace-parallel-plan-strips-lt-length-plan-plus-one-then:
assumes length (trace-parallel-plan-strips \(I \pi\) ) \(<\) length \(\pi+1\)
shows \(\neg\) are-all-operators-applicable
(execute-parallel-plan I (take (length (trace-parallel-plan-strips \(I \pi)-1) \pi)\) )
( \(\pi\) ! (length (trace-parallel-plan-strips \(I \pi)-1)\) )
\(\vee \neg\) are-all-operator-effects-consistent ( \(\pi\) ! (length (trace-parallel-plan-strips \(I \pi\) )
- 1))
using assms
proof (induction \(\pi\) arbitrary: \(I\) )
case (Cons ops \(\pi\) )
let \(? \tau=\) trace-parallel-plan-strips \(I(\) ops \(\# \pi)\)
and \(? I^{\prime}=\) execute-parallel-operator I ops
show ?case
proof (cases are-all-operators-applicable I ops \(\wedge\) are-all-operator-effects-consistent
```

ops)

```
case True
then have \(\tau\)-is: ? \(\tau=I \#\) trace-parallel-plan-strips \({ }^{2} I^{\prime} \pi\)
    by fastforce
show ?thesis
    proof (cases length (trace-parallel-plan-strips ? \(\left.I^{\prime} \pi\right)<\) length \(\pi+1\) )
        case True
        then have \(\neg\) are-all-operators-applicable
                (execute-parallel-plan? ? \({ }^{\prime}\)
                (take (length (trace-parallel-plan-strips ? \(\left.\left.\left.\left.I^{\prime} \pi\right)-1\right) \pi\right)\right)\)
            \(\left(\pi!\left(\right.\right.\) length \(\left(\right.\) trace-parallel-plan-strips ? \(\left.\left.\left.I^{\prime} \pi\right)-1\right)\right)\)
            \(\vee \neg\) are-all-operator-effects-consistent
            \(\left(\pi!\left(\right.\right.\) length \(\left(\right.\) trace-parallel-plan-strips ? \(\left.\left.\left.I^{\prime} \pi\right)-1\right)\right)\)
            using Cons.IH[of ?I \(]\)
            by blast
            moreover have trace-parallel-plan-strips ? \(I^{\prime} \pi \neq[]\)
            using trace-parallel-plan-strips-not-nil
            by blast
            ultimately show ?thesis
                unfolding take-Cons'
                by simp
    next
        case False
        then have length (trace-parallel-plan-strips ? \(\left.I^{\prime} \pi\right) \geq\) length \(\pi+1\)
            by fastforce
        thm Cons.prems
        moreover have length (trace-parallel-plan-strips I (ops \# \(\pi\) ) )
            \(=1+\) length (trace-parallel-plan-strips ? \(\left.I^{\prime} \pi\right)\)
            using True
            by force
        moreover have length (trace-parallel-plan-strips ? \(I^{\prime} \pi\) )
            \(<\) length (ops \# \(\pi\) )
            using Cons.prems calculation(2)
            by force
        ultimately have False
            by fastforce
        thus ?thesis..
    qed
next
    case False
    then have \(\tau\)-is-singleton: ? \(\tau=[I]\)
        using False
        by auto
    then have ops \(=(\) ops \(\# \pi)!(\) length ? \(\tau-1)\)
    by fastforce
    moreover have execute-parallel-plan \(I(\) take (length ? \(\tau-1) \pi)=I\)
    using \(\tau\)-is-singleton
    by auto
    - TODO slow.
```

```
            ultimately show ?thesis
            using False
            by auto
    qed
qed simp
lemma trace-parallel-plan-step-effect-is:
    assumes }k<\mathrm{ length (trace-parallel-plan-strips I }\pi\mathrm{ ) - 1
    shows trace-parallel-plan-strips I \pi!Suc k
    = execute-parallel-operator (trace-parallel-plan-strips I }\pi!k)(\pi!k
    proof -
    - NOTE Rewrite the proposition using lemma trace-parallel-plan-strips-subplan.
    {
        let ? }\tau=\mathrm{ trace-parallel-plan-strips I }
        have Suc k < length ?\tau
            using assms
            by linarith
        hence trace-parallel-plan-strips I \pi!Suc k
            = execute-parallel-plan I (take (Suc k)\pi)
            using trace-parallel-plan-plan-prefix[of Suc k I \pi]
            by blast
    }
    moreover have execute-parallel-plan I (take (Suc k) \pi)
        = execute-parallel-operator (trace-parallel-plan-strips I \pi!k) (\pi!k)
        using assms
        proof (induction k arbitrary: I \pi)
            case 0
            then have execute-parallel-operator (trace-parallel-plan-strips I \pi!0) (\pi!
0)
                = execute-parallel-operator I ( }\pi!0
                using trace-parallel-plan-strips-head-is-initial-state[of I \pi]
                by argo
            moreover {
            obtain ops }\mp@subsup{\pi}{}{\prime}\mathrm{ where }\pi=\mathrm{ ops # }\mp@subsup{\pi}{}{\prime
            using plan-is-at-least-singleton-plan-if-trace-has-at-least-two-elements[OF
0.prems]
            by blast
            then have take (Suc 0) }\pi=[\pi!0
                by simp
            hence execute-parallel-plan I (take (Suc 0) \pi)
                = execute-parallel-plan I [\pi!0]
            by argo
        }
        moreover {
            have 0 < length (trace-parallel-plan-strips I \pi) - 1
                using trace-parallel-plan-strips-length-gt-one-if 0.prems
                by fastforce
            hence are-all-operators-applicable I ( }\pi!0
                \wedge ~ a r e - a l l - o p e r a t o r - e f f e c t s - c o n s i s t e n t ~ ( ~ \pi ! 0 ) ~
```

```
            using trace-parallel-plan-strips-operator-preconditions[of 0 I 
                trace-parallel-plan-strips-head-is-initial-state[of I \pi]
            by argo
    }
    ultimately show ?case
    by auto
next
    case (Suc k)
    obtain ops }\mp@subsup{\pi}{}{\prime}\mathrm{ where }\pi\mathrm{ -split: }\pi=\mathrm{ ops # }\mp@subsup{\pi}{}{\prime
        using plan-is-at-least-singleton-plan-if-trace-has-at-least-two-elements[OF
Suc.prems]
        by blast
    let ? I' = execute-parallel-operator I ops
    {
        have length (trace-parallel-plan-strips I \pi) =
            1 + length (trace-parallel-plan-strips ? I' }\mp@subsup{\pi}{}{\prime}\mathrm{ )
            using Suc.prems \pi}\pi\mathrm{ -split
            by fastforce
        then have k< length (trace-parallel-plan-strips ?I' }\mp@subsup{\pi}{}{\prime}\mathrm{ )
            using Suc.prems
            by fastforce
        moreover have trace-parallel-plan-strips I \pi!Suc k
            = trace-parallel-plan-strips ?I'}\mp@subsup{\pi}{}{\prime}!
            using Suc.prems \pi-split
            by force
        ultimately have trace-parallel-plan-strips I \pi!Suc k
            = execute-parallel-plan ?I' (take k }\mp@subsup{\pi}{}{\prime}\mathrm{ )
            using trace-parallel-plan-plan-prefix[of k ? 'I' \pi}
            by argo
    }
    moreover have execute-parallel-plan I (take (Suc (Suc k)) \pi)
        = execute-parallel-plan ?I' (take (Suc k) \pi')
        using Suc.prems \pi-split
        by fastforce
    moreover {
        have 0<length (trace-parallel-plan-strips I \pi) - 1
            using Suc.prems
            by linarith
    hence are-all-operators-applicable I ( }\pi!0
            \are-all-operator-effects-consistent ( }\pi!0
            using trace-parallel-plan-strips-operator-preconditions[of 0 I 
                trace-parallel-plan-strips-head-is-initial-state[of I \pi]
            by argo
    }
    ultimately show ?case
        using Suc.IH Suc.prems \pi}\pi\mathrm{ -split
        by auto
    qed
ultimately show ?thesis
```

```
        using assms
        by argo
    qed
```

- Show that every state in a plan execution trace of a valid problem description is defined for all problem variables. This is true because the initial state is defined for all problem variables-by definition of is-valid-problem-strips $\Pi$-and no operator can remove a previously defined variable (only positive and negative effects are possible).
lemma trace-parallel-plan-strips-none-if:
fixes $\Pi$ :: 'a strips-problem
assumes is-valid-problem-strips $\Pi$
and is-parallel-solution-for-problem $\Pi \pi$
and $k<$ length (trace-parallel-plan-strips $\left.\left((\Pi)_{I}\right) \pi\right)$
shows (trace-parallel-plan-strips $\left.\left((\Pi)_{I}\right) \pi!k\right) v=$ None $\longleftrightarrow v \notin \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)$
proof -
let ?vs $=$ strips-problem.variables-of $\Pi$
and ${ }^{2}$ ops $=$ strips-problem.operators-of $\Pi$
and ${ }^{2} \tau=$ trace-parallel-plan-strips $\left((\Pi)_{I}\right) \pi$
and ?I = strips-problem.initial-of $\Pi$
show ?thesis
using assms
proof (induction $k$ )
case 0
have ? $\tau!0=? I$
using trace-parallel-plan-strips-head-is-initial-state
by auto
then show ?case
using is-valid-problem-strips-initial-of-dom[OF assms(1)]
by auto
next
case (Suc k)
have $k$-lt-length- $\tau$-minus-one: $k<$ length ? $\tau-1$
using Suc.prems(3)
by linarith
then have IH: (trace-parallel-plan-strips ? $I \pi!k) v=$ None $\longleftrightarrow v \notin$ set
((П) $\mathcal{V})$
using Suc.IH[OF Suc.prems(1, 2)]
by force
have $\tau$-Suc-k-is: $(? \tau!$ Suc $k)=$ execute-parallel-operator $(? \tau!k)(\pi!k)$ using trace-parallel-plan-step-effect-is[OF $k$-lt-length- $\tau$-minus-one].
have all-operators-applicable: are-all-operators-applicable $(? \tau!k)(\pi!k)$ and all-effects-consistent: are-all-operator-effects-consistent $(\pi!k)$
using trace-parallel-plan-strips-operator-preconditions[OF $k$-lt-length- $\tau$-minus-one] by $\operatorname{simp}+$
show ?case
proof (rule iffI)
assume $\tau$-Suc-k-of-v-is-None: (? $\tau!$ Suc $k) v=$ None

```
    show v\not\in set ((\Pi)\mathcal{V})
    proof (rule ccontr)
        assume }\negv\not\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{V}}{)
        then have v-in-set-vs:v\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{V}}{})
            by blast
        show False
            proof (cases \existsop\in set ( }\pi!k)\mathrm{ .
            v\in\operatorname{set (add-effects-of op)}\veev\in\operatorname{set (delete-effects-of op))}
            case True
            then obtain op
                where op-in-\mp@subsup{\pi}{k}{}:op\in\operatorname{set}(\pi!k)
                    and}v\inset (add-effects-of op) \veev\in set (delete-effects-of op)..
            then consider (A)v\in set (add-effects-of op)
                | (B) v\in set (delete-effects-of op)
                by blast
            thus False
                using execute-parallel-operator-positive-effect-if[OF
                    all-operators-applicable all-effects-consistent op-in-\pi
                                    execute-parallel-operator-negative-effect-if [OF
                                    all-operators-applicable all-effects-consistent op-in-\pi}\mp@subsup{\pi}{k}{}
                                    \tau - S u c - k - o f - v - i s - N o n e ~ \tau - S u c - k - i s
                by (cases, fastforce+)
        next
            case False
            then have }\forallop\in\operatorname{set}(\pi!k)
                v\not\in set (add-effects-of op) ^v\not\in set (delete-effects-of op)
                    by blast
            then have (?\tau ! Suc k) v = (?\tau!k)v
                using execute-parallel-operator-no-effect-if }\tau\mathrm{ -Suc-k-is
                    by fastforce
            then have v\not\in set ((\Pi)\mathcal{V})
                    using IH \tau-Suc-k-of-v-is-None
                    by simp
            thus False
                using v-in-set-vs
                by blast
        qed
    qed
next
    assume v-notin-vs: v\not\in set ((\Pi)\mathcal{V})
{
    fix op
    assume op-in-\mp@subsup{\pi}{k}{}:op \in set (\pi!k)
    {
        have }1<length?
    using trace-parallel-plan-strips-length-gt-one-if[OF k-lt-length-\tau-minus-one].
        then have 0<length ? \tau - 1
            using k-lt-length-\tau-minus-one
            by linarith
```

```
    moreover have length ? \tau - 1 \leq length \pi
                using length-trace-parallel-plan-strips-lte-length-plan-plus-one
le-diff-conv
            by blast
            then have k< length \pi
            using k-lt-length-\tau-minus-one
            by force
            hence }\pi!k\in\mathrm{ set }
            by simp
    }
    then have op-in-ops:op \in set ?ops
            using is-parallel-solution-for-problem-operator-set[OF assms(2) -
op-in-\pi}\mp@subsup{\pi}{k}{
            by force
            hence v\not\in set (add-effects-of op) and v}\not=\mathrm{ set (delete-effects-of op)
                subgoal
                            using is-valid-problem-strips-operator-variable-sets(2) assms(1)
op-in-ops
                v-notin-vs
                    by auto
            subgoal
                using is-valid-problem-strips-operator-variable-sets(3) assms(1)
op-in-ops
                v-notin-vs
                    by auto
                    done
            }
            then have (?\tau!Suc k) v = (?\tau!k)v
                    using execute-parallel-operator-no-effect-if }\tau\mathrm{ -Suc-k-is
                    by metis
            thus (?\tau!Suc k) v = None
                using IH v-notin-vs
                by fastforce
            qed
        qed
    qed
```

Finally, given initial and goal states $I$ and $G$, we can show that it's equivalent to say that $\pi$ is a solution for $I$ and $G$-i.e. $G \subseteq_{m}$ execute-parallel-plan $I$ $\pi$-and that the goal state is subsumed by the last element of the trace of $\pi$ with initial state $I$.
lemma execute-parallel-plan-reaches-goal-iff-goal-is-last-element-of-trace:
$G \subseteq_{m}$ execute-parallel-plan I $\pi$
$\longleftrightarrow G \subseteq_{m}$ last (trace-parallel-plan-strips $I \pi$ )
proof -
let $? L H S=G \subseteq_{m}$ execute-parallel-plan $I \pi$
and ?RHS $=G \subseteq_{m}$ last (trace-parallel-plan-strips $I \pi$ )
show ?thesis
proof (rule iffI)

```
assume ?LHS
thus ?RHS
    proof (induction \pi arbitrary:I)
        - NOTE Nil case follows from simplification.
        case (Cons a \pi)
        thus ?case
            using Cons.prems
    proof (cases are-all-operators-applicable I a ^ are-all-operator-effects-consistent
a)
            case True
            let ?I' = execute-parallel-operator I a
            {
                        have execute-parallel-plan I (a#\pi)= execute-parallel-plan ? I' }
                using True
                by auto
                    then have G}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan ?I' }
                    using Cons.prems
                    by presburger
                    hence G \subseteqm
                    using Cons.IH[of ?I']
                by blast
            }
            moreover {
                    have trace-parallel-plan-strips I (a# \pi)
                    = I# trace-parallel-plan-strips ? 'I'}
                    using True
                by simp
                    then have last (trace-parallel-plan-strips I (a#\pi))
                    = last (I # trace-parallel-plan-strips ?I' }\pi\mathrm{ )
                    by argo
                    hence last (trace-parallel-plan-strips I (a#\pi))
                        = last (trace-parallel-plan-strips ?I' \pi)
                using trace-parallel-plan-strips-last-cons-then[of I ?I' \pi]
                    by argo
            }
            ultimately show ?thesis
                by argo
            qed force
        qed simp
next
    assume ?RHS
    thus ?LHS
        proof (induction \pi arbitrary:I)
            - NOTE Nil case follows from simplification.
            case (Cons a \pi
            thus ?case
                            proof (cases are-all-operators-applicable I a ^ are-all-operator-effects-consistent
a)
                    case True
```

```
                    let ?I' = execute-parallel-operator I a
                    {
        have trace-parallel-plan-strips I (a#\pi)=I# (trace-parallel-plan-strips
?I'}\pi
            using True
            by simp
                    then have last (trace-parallel-plan-strips I (a#\pi))
                        = last (trace-parallel-plan-strips ?I' }\pi\mathrm{ )
                        using trace-parallel-plan-strips-last-cons-then[of I ?I' \pi]
                        by argo
                            hence G}\mp@subsup{\subseteq}{m}{}\mathrm{ last (trace-parallel-plan-strips ? ?'' }\pi\mathrm{ )
                            using Cons.prems
                            by argo
                    }
                        thus ?thesis
                    using True Cons
                    by simp
                    next
                        case False
                        then have last (trace-parallel-plan-strips I (a#\pi)) =I
                        and execute-parallel-plan I (a# \pi) = I
                        by (fastforce, force)
                    thus ?thesis
                    using Cons.prems
                    by argo
            qed
        qed fastforce
    qed
qed
```


### 3.3 Serializable Parallel Plans

With the groundwork on parallel and serial execution of STRIPS in place we can now address the question under which conditions a parallel solution to a problem corresponds to a serial solution and vice versa. As we will see (in theorem ??), while a serial plan can be trivially rewritten as a parallel plan consisting of singleton operator list for each operator in the plan, the condition for parallel plan solutions also involves non interference.

```
lemma execute-parallel-operator-equals-execute-sequential-strips-if:
    fixes \(s::\) ('variable, bool) state
    assumes are-all-operators-applicable s ops
        and are-all-operator-effects-consistent ops
        and are-all-operators-non-interfering ops
    shows execute-parallel-operator sops \(=\) execute-serial-plan sops
    using assms
    proof (induction ops arbitrary: s)
    case Nil
    have execute-parallel-operator s Nil
```

```
    = foldl (++)s(map (map-of ○ effect-to-assignments) Nil)
    using Nil.prems(1,2)
    unfolding execute-parallel-operator-def
    by presburger
    also have ... = s
    by simp
    finally have execute-parallel-operator s Nil =s
        by blast
    moreover have execute-serial-plan s Nil =s
    by auto
    ultimately show ?case
        by simp
next
    case (Cons a ops)
    - NOTE Use the preceding lemmas to show that the premises hold for the
sublist and use the IH to obtain the theorem for the sublist ops.
    have a: is-operator-applicable-in s a
        using are-all-operators-applicable-cons Cons.prems(1)
        by blast+
    let ? s' = s++ map-of (effect-to-assignments a)
    {
        from Cons.prems
        have are-all-operators-applicable ?s' ops
            and are-all-operator-effects-consistent ops
            and are-all-operators-non-interfering ops
            using execute-parallel-plan-precondition-cons
            by blast+
        then have execute-serial-plan ?s' ops
            = execute-parallel-operator ? s' ops
            using Cons.IH
            by presburger
    }
    moreover from Cons.prems
    have execute-parallel-operators (Cons a ops)
        = execute-parallel-operator ?s' ops
        using execute-parallel-operator-cons-equals-corollary
        unfolding execute-operator-def
        by simp
    moreover
    from a have execute-serial-plan s (Cons a ops)
        = execute-serial-plan ?s' ops
        unfolding execute-serial-plan-def execute-operator-def
            is-operator-applicable-in-def
        by fastforce
    ultimately show ?case
        by argo
qed
lemma execute-serial-plan-split-i:
```

```
assumes are-all-operators-applicable s (op \# \(\pi\) )
    and are-all-operators-non-interfering (op \(\# \pi\) )
shows are-all-operators-applicable \((s \gg o p) \pi\)
using assms
proof (induction \(\pi\) arbitrary: \(s\) )
    case Nil
    then show ?case
        unfolding are-all-operators-applicable-def
    by \(\operatorname{simp}\)
next
    case (Cons op \({ }^{\prime} \pi\) )
    let ? \(t=s \gg o p\)
    \{
        fix \(x\)
        assume \(x \in \operatorname{set}\left(o p^{\prime} \# \pi\right)\)
        moreover have \(o p \in \operatorname{set}\left(o p \# o p^{\prime} \# \pi\right)\)
        by \(\operatorname{simp}\)
    moreover have \(\neg\) are-operators-interfering op \(x\)
        using Cons.prems(2) calculation(1)
        unfolding are-all-operators-non-interfering-def list-all-iff
        by fastforce
    moreover have is-operator-applicable-in sop
            using Cons.prems(1)
            unfolding are-all-operators-applicable-def list-all-iff
                is-operator-applicable-in-def
        by force
    moreover have is-operator-applicable-in s x
    using are-all-operators-applicable-cons(2)[OF Cons.prems(1)] calculation(1)
            unfolding are-all-operators-applicable-def list-all-iff
                is-operator-applicable-in-def
            by fast
    ultimately have is-operator-applicable-in ?t \(x\)
            using execute-parallel-plan-precondition-cons-i[of op \(x\) s \(]\)
            by (auto simp: execute-operator-def)
\}
    thus ?case
    using are-all-operators-applicable-cons(2)
    unfolding is-operator-applicable-in-def
        STRIPS-Representation.is-operator-applicable-in-def
        are-all-operators-applicable-def list-all-iff
    by \(\operatorname{simp}\)
qed
```

- Show that plans $\pi$ can be split into separate executions of partial plans $\pi_{1}$ and $\pi_{2}$ with $\pi=\pi_{1} @ \pi_{2}$, if all operators in $\pi_{1}$ are applicable in the given state $s$ and there is no interference between subsequent operators in $\pi_{1}$. This is the case because non interference ensures that no precondition for any operator in $\pi_{1}$ is negated by the execution of a preceding operator. Note that the non interference constraint excludes partial plans where a precondition is first violated during execution but later
restored which would also allow splitting but does not meet the non interference constraint (which must hold for all possible executing orders)
lemma execute-serial-plan-split:
fixes $s::$ ('variable, bool) state
assumes are-all-operators-applicable s $\pi_{1}$
and are-all-operators-non-interfering $\pi_{1}$
shows execute-serial-plans $\left(\pi_{1} @ \pi_{2}\right)$
$=$ execute-serial-plan (execute-serial-plan $\left.s \pi_{1}\right) \pi_{2}$
using assms
proof (induction $\pi_{1}$ arbitrary: $s$ )
case (Cons op $\pi_{1}$ )
let ? $t=s \gg o p$
\{
have are-all-operators-applicable $(s \gg o p) \pi_{1}$ using execute-serial-plan-split-i[OF Cons.prems(1, 2)]. moreover have are-all-operators-non-interfering $\pi_{1}$ using are-all-operators-non-interfering-tail[OF Cons.prems(2)].
ultimately have execute-serial-plan ?t $\left(\pi_{1} @ \pi_{2}\right)=$ execute-serial-plan (execute-serial-plan ?t $\pi_{1}$ ) $\pi_{2}$
using Cons.IH[of ? t]
by blast
\}
moreover have STRIPS-Representation.is-operator-applicable-in s op using Cons.prems(1)
unfolding are-all-operators-applicable-def list-all-iff by fastforce
ultimately show ?case
unfolding execute-serial-plan-def
by $\operatorname{simp}$
qed $\operatorname{simp}$
lemma embedding-lemma- $i$ :
fixes $I$ :: ('variable, bool) state
assumes is-operator-applicable-in I op
and are-operator-effects-consistent op op
shows $I \gg o p=$ execute-parallel-operator $I[o p]$
proof -
have are-all-operators-applicable I [op]
using assms(1)
unfolding are-all-operators-applicable-def list-all-iff is-operator-applicable-in-def by fastforce
moreover have are-all-operator-effects-consistent [op]
unfolding are-all-operator-effects-consistent-def list-all-iff
using assms(2)
by fastforce
moreover have are-all-operators-non-interfering [op]
by $\operatorname{simp}$
moreover have $I \gg o p=$ execute-serial-plan $I[o p]$
using assms(1)
unfolding is-operator-applicable-in-def
by (simp add: assms(1) execute-operator-def)
ultimately show ?thesis
using execute-parallel-operator-equals-execute-sequential-strips-if by force
qed
lemma execute-serial-plan-is-execute-parallel-plan-ii:
fixes $I$ :: 'variable strips-state
assumes $\forall$ op $\in$ set $\pi$. are-operator-effects-consistent op op
and $G \subseteq_{m}$ execute-serial-plan $I \pi$
shows $G \subseteq_{m}$ execute-parallel-plan I (embed $\pi$ )
proof -
show ?thesis
using assms
proof (induction $\pi$ arbitrary: $I$ )
case (Cons op $\pi$ )
then show ?case
proof (cases is-operator-applicable-in I op) case True
let ? $J=I \gg o p$
and $? J^{\prime}=$ execute-parallel-operator $I[o p]$ \{
have $G \subseteq_{m}$ execute-serial-plan ? $J \pi$ using Cons.prems(2) True unfolding is-operator-applicable-in-def by (simp add: True)
hence $G \subseteq_{m}$ execute-parallel-plan ?J (embed $\pi$ )
using Cons.IH[of ? J] Cons.prems(1)
by fastforce
\}
moreover \{
have are-all-operators-applicable I [op]
using True
unfolding are-all-operators-applicable-def list-all-iff is-operator-applicable-in-def by fastforce
moreover have are-all-operator-effects-consistent [op]
unfolding are-all-operator-effects-consistent-def list-all-iff using Cons.prems(1)
by fastforce
moreover have ? $J=$ ? $J^{\prime}$
using execute-parallel-operator-equals-execute-sequential-strips-if $[O F$
calculation(1, 2)] Cons.prems(1) True
unfolding is-operator-applicable-in-def by ( simp add: True)
ultimately have execute-parallel-plan I (embed (op \# $\pi$ ) )
$=$ execute-parallel-plan? $J($ embed $\pi)$

```
                    by fastforce
                }
                        ultimately show ?thesis
                        by presburger
            next
            case False
            then have G}\mp@subsup{\subseteq}{m}{}
                    using Cons.prems is-operator-applicable-in-def
                    by simp
                    moreover {
                    have \negare-all-operators-applicable I [op]
                using False
                    unfolding are-all-operators-applicable-def list-all-iff
                    is-operator-applicable-in-def
                by force
                    hence execute-parallel-plan I (embed (op # \pi)) = I
                by simp
                    }
                    ultimately show ?thesis
                by presburger
            qed
        qed simp
    qed
lemma embedding-lemma-iii:
    fixes \Pi:: 'a strips-problem
    assumes }\forallop\in\mathrm{ set }\pi\mathrm{ . op }\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{}
    shows }\forallops\in\operatorname{set}(embed \pi).\forallop\in set ops. op \in set ((\Pi\mp@subsup{)}{\mathcal{O}}{}
    proof -
    have nb: set (embed \pi)={[op]|op.op\in set \pi }
            by (induction \pi; force)
    {
        fix ops
        assume ops \in set (embed \pi)
        moreover obtain op where op fet \pi and ops = [op]
            using nb calculation
            by blast
            ultimately have }\forallop\in\mathrm{ set ops. op }\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{}
                using assms(1)
                by simp
    }
    thus ?thesis..
    qed
```

We show in the following theorem that-as mentioned-a serial solution $\pi$ to a STRIPS problem $\Pi$ corresponds directly to a parallel solution obtained by embedding each operator in $\pi$ in a list (by use of function List-Supplement.embed). The proof shows this by first confirming that

$$
\begin{aligned}
G & \subseteq_{m} \text { execute-serial-plan }\left((\Pi)_{I}\right) \pi \\
& \Longrightarrow G \subseteq_{m} \text { execute-serial-plan }\left((\Pi)_{I}\right)(\text { embed } \pi)
\end{aligned}
$$

using lemma ; and moreover by showing that

$$
\forall o p s \in \operatorname{set}(\text { embed } \pi) . \forall o p \in \text { set ops. op } \in(\Pi)_{\mathcal{O}}
$$

meaning that under the given assumptions, all parallel operators of the embedded serial plan are again operators in the operator set of the problem.
theorem embedding-lemma:
assumes is-valid-problem-strips $\Pi$
and is-serial-solution-for-problem $\Pi \pi$
shows is-parallel-solution-for-problem $\Pi$ (embed $\pi$ )
proof -
have $n b_{1}: \forall o p \in \operatorname{set} \pi$. op $\in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)$
using assms(2)
unfolding is-serial-solution-for-problem-def list-all-iff ListMem-iff opera-tors-of-def
by blast
\{
fix $o p$
assume $o p \in$ set $\pi$
moreover have $o p \in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)$
using $n b_{1}$ calculation
by fast
moreover have is-valid-operator-strips $\Pi$ op
using assms(1) calculation(2)
unfolding is-valid-problem-strips-def is-valid-problem-strips-def list-all-iff operators-of-def
by meson
moreover have list-all $(\lambda v . \neg L i s t M e m v($ delete-effects-of op)) (add-effects-of
op)
and list-all $(\lambda v . \neg$ ListMem $v$ (add-effects-of op)) (delete-effects-of op)
using calculation(3)
unfolding is-valid-operator-strips-def
by meson+
moreover have $\neg l i s t-e x(\lambda v$. ListMem $v$ (delete-effects-of op)) (add-effects-of
op)
and $\neg l i s t-e x(\lambda v$. ListMem $v$ (add-effects-of op)) (delete-effects-of op)
using calculation $(4,5)$ not-list-ex-equals-list-all-not
by blast+
moreover have $\neg$ list-ex $(\lambda v$. list-ex $((=) v)$ (delete-effects-of op)) (add-effects-of
op)
and $\neg$ list-ex $(\lambda v$. list-ex $((=) v)($ add-effects-of op)) (delete-effects-of op)
using calculation ( 6,7 )
unfolding list-ex-iff ListMem-iff
by blast+
ultimately have are-operator-effects-consistent op op

```
        unfolding are-operator-effects-consistent-def Let-def
        by blast
    } note n\mp@subsup{b}{2}{}= this
    moreover {
    have (\Pi)}\mp@subsup{G}{G}{}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-serial-plan ((П)
        using assms(2)
        unfolding is-serial-solution-for-problem-def
        by simp
    hence }(\Pi\mp@subsup{)}{G}{}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan ((П) I) (embed }\pi\mathrm{ )
        using execute-serial-plan-is-execute-parallel-plan-ii nb 
        by blast
    }
    moreover have }\forallops\in\mathrm{ set (embed }\pi).\forallop\in set ops. op \in set ((\Pi) ( ) )
    using embedding-lemma-iii[OF nb [].
    ultimately show ?thesis
    unfolding is-parallel-solution-for-problem-def goal-of-def
        initial-of-def operators-of-def list-all-iff ListMem-iff
    by blast
qed
lemma flattening-lemma-i:
    fixes П:: 'a strips-problem
    assumes }\forall\mathrm{ ops }\in\mathrm{ set }\pi.\forallop\in set ops. op \in set ((\Pi) ( ) )
    shows }\forallop\in\operatorname{set}(\mathrm{ concat }\pi).op\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{}
    proof -
        {
            fix op
            assume op \in set (concat \pi)
            moreover have op (\bigcupops \in set \pi}\mathrm{ . set ops)
                using calculation
                unfolding set-concat.
            then obtain ops where ops \in set \pi}\mathrm{ and op f set ops
                using UN-iff
                by blast
            ultimately have op \in set ((\Pi)}\mp@subsup{)}{\mathcal{O}}{}
            using assms
            by blast
    }
    thus ?thesis..
    qed
lemma flattening-lemma-ii:
    fixes I :: 'variable strips-state
    assumes }\forallops\in\mathrm{ set }\pi\mathrm{ . ヨop.ops = [op]^ is-valid-operator-strips П op
    and G\subseteq}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan I }
    shows G}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-serial-plan I (concat }\pi\mathrm{ )
proof -
    let ? }\mp@subsup{\pi}{}{\prime}=\mathrm{ concat }
```


## \{

fix $o p$
assume is-valid-operator-strips $\Pi$ op
moreover have list-all ( $\lambda v$. $\neg$ ListMem $v$ (delete-effects-of op)) (add-effects-of
op)
and list-all ( $\lambda v$. $\neg$ ListMem $v$ (add-effects-of op)) (delete-effects-of op)
using calculation(1)
unfolding is-valid-operator-strips-def
by meson +
moreover have $\neg$ list-ex ( $\lambda v$. ListMem $v$ (delete-effects-of op)) (add-effects-of
op)
and $\neg$ list-ex ( $\lambda v$. ListMem $v($ add-effects-of op)) (delete-effects-of op)
using calculation $(2,3)$ not-list-ex-equals-list-all-not
by blast+
moreover have $\neg$ list-ex $(\lambda v$. list-ex $((=) v)$ (delete-effects-of op)) (add-effects-of
op)
and $\neg l i s t-e x(\lambda v$. list-ex $((=) v)($ add-effects-of $o p))($ delete-effects-of op)
using calculation (4, 5)
unfolding list-ex-iff ListMem-iff
by blast+
ultimately have are-operator-effects-consistent op op
unfolding are-operator-effects-consistent-def Let-def
by blast
$\}$ note $n b_{1}=$ this
show ?thesis
using assms
proof (induction $\pi$ arbitrary: $I$ )
case (Cons ops $\pi$ )
obtain $o p$ where ops-is: ops $=[o p]$ and is-valid-op: is-valid-operator-strips
$\Pi$ ор
using Cons.prems(1)
by fastforce
show ?case
proof (cases are-all-operators-applicable I ops)
case True
let ? $J=$ execute-parallel-operator $I[o p]$
and $? J^{\prime}=I \gg o p$
have $n b_{2}$ : is-operator-applicable-in I op
using True ops-is
unfolding are-all-operators-applicable-def list-all-iff
is-operator-applicable-in-def
by $\operatorname{simp}$
have $n b_{3}$ : are-operator-effects-consistent op op using $n b_{1}[$ OF is-valid-op $]$.

## \{

then have are-all-operator-effects-consistent ops unfolding are-all-operator-effects-consistent-def list-all-iff using ops-is
by fastforce

```
                    hence G\subseteqm}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan ?J }
                    using Cons.prems(2) ops-is True
                    by fastforce
    }
    moreover have execute-serial-plan I (concat (ops # \pi))
            = execute-serial-plan ? J' (concat \pi)
            using ops-is nb
            unfolding is-operator-applicable-in-def
            by (simp add: execute-operator-def n\mp@subsup{b}{2}{}}\mathrm{ )
            moreover have ? J = ? J'
            unfolding execute-parallel-operator-def execute-operator-def comp-apply
                    by fastforce
            ultimately show ?thesis
            using Cons.IH Cons.prems
            by force
        next
            case False
            moreover have G\subseteqm}
                    using Cons.prems(2) calculation
            by force
            moreover {
                have \negis-operator-applicable-in I op
                    using ops-is False
                    unfolding are-all-operators-applicable-def list-all-iff
                    is-operator-applicable-in-def
                by fastforce
                    hence execute-serial-plan I (concat (ops # \pi)) = I
                    using ops-is is-operator-applicable-in-def
                    by simp
            }
            ultimately show ?thesis
                by argo
            qed
        qed force
qed
```

The opposite direction is also easy to show if we can normalize the parallel plan to the form of an embedded serial plan as shown below.

```
lemma flattening-lemma:
    assumes is-valid-problem-strips \(\Pi\)
        and \(\forall o p s \in\) set \(\pi\). \(\exists o p\). ops \(=[o p]\)
        and is-parallel-solution-for-problem \(\Pi \pi\)
    shows is-serial-solution-for-problem \(\Pi\) (concat \(\pi\) )
    proof -
    let \(? \pi^{\prime}=\) concat \(\pi\)
    \{
        have \(\forall\) ops \(\in\) set \(\pi . \forall o p \in\) set ops. op \(\in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)\)
            using assms(3)
            unfolding is-parallel-solution-for-problem-def list-all-iff ListMem-iff
```

```
            by force
        hence }\forallop\in\operatorname{set}?\mp@subsup{\pi}{}{\prime}.op\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{}
            using flattening-lemma-i
            by blast
}
moreover {
    {
        fix ops
        assume ops \in set \pi
        moreover obtain op where ops = [op]
            using assms(2) calculation
            by blast
        moreover have op \in set ((\Pi)
            using assms(3) calculation
            unfolding is-parallel-solution-for-problem-def list-all-iff ListMem-iff
            by force
        moreover have is-valid-operator-strips \Pi op
            using assms(1) calculation(3)
            unfolding is-valid-problem-strips-def Let-def list-all-iff ListMem-iff
            by simp
        ultimately have \existsop.ops = [op]^is-valid-operator-strips \Pi op
            by blast
        }
    moreover have (\Pi) }\mp@subsup{)}{G}{}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan ((П)}\mp@subsup{)}{I}{})
        using assms(3)
        unfolding is-parallel-solution-for-problem-def
        by simp
    ultimately have }(\Pi\mp@subsup{)}{G}{}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-serial-plan ((П)
        using flattening-lemma-ii
        by blast
}
ultimately show is-serial-solution-for-problem \Pi ? |'
    unfolding is-serial-solution-for-problem-def list-all-iff ListMem-iff
    by simp
qed
```

Finally, we can obtain the important result that a parallel plan with a trace that reaches the goal state of a given problem $\Pi$, and for which both the parallel operator execution condition as well as non interference is assured at every point $k<$ length $\pi$, the flattening of the parallel plan concat $\pi$ is a serial solution for the initial and goal state of the problem. To wit, by lemma ?? we have

```
\(\left(G \subseteq_{m}\right.\) execute-parallel-plan \(\left.I \pi\right)\)
    \(=\left(G \subseteq_{m}\right.\) last (trace-parallel-plan-strips \(\left.\left.I \pi\right)\right)\)
```

so the second assumption entails that $\pi$ is a solution for the initial state and the goal state of the problem. (which implicitely means that $\pi$ is a solution for the inital state and goal state of the problem). The trace formulation
is used in this case because it allows us to write the - state dependentapplicability condition more succinctly. The proof (shown below) is by structural induction on $\pi$ with arbitrary initial state.

```
theorem execute-parallel-plan-is-execute-sequential-plan-if:
    fixes \(I\) :: ('variable, bool) state
    assumes is-valid-problem \(\Pi\)
        and \(G \subseteq_{m}\) last (trace-parallel-plan-strips \(I \pi\) )
        and \(\forall k<\) length \(\pi\).
        are-all-operators-applicable (trace-parallel-plan-strips \(I \pi!k)(\pi!k)\)
        \(\wedge\) are-all-operator-effects-consistent \((\pi!k)\)
        \(\wedge\) are-all-operators-non-interfering \((\pi!k)\)
    shows \(G \subseteq_{m}\) execute-serial-plan I (concat \(\pi\) )
    using assms
    proof (induction \(\pi\) arbitrary: \(I\) )
        case (Cons ops \(\pi\) )
        let ?ops \({ }^{\prime}=\) take \((\) length ops \()(\) concat \((\) ops \(\# \pi))\)
        let ? \(J=\) execute-parallel-operator I ops
            and ? \(J^{\prime}=\) execute-serial-plan I ?ops \({ }^{\prime}\)
        \{
        have trace-parallel-plan-strips \(I \pi!0=I\) and (ops \(\# \pi)!0=o p s\)
        unfolding trace-parallel-plan-strips-head-is-initial-state
        by \(\operatorname{simp}+\)
    then have are-all-operators-applicable I ops
        and are-all-operator-effects-consistent ops
        and are-all-operators-non-interfering ops
        using Cons.prems(3)
        by auto+
        then have trace-parallel-plan-strips \(I\) (ops \(\# \pi\) )
        \(=I \#\) trace-parallel-plan-strips ?J \(\pi\)
        by fastforce
        \} note \(n b=t h i s\)
        \{
        have last (trace-parallel-plan-strips I (ops \# \(\pi\) ))
        \(=\) last (trace-parallel-plan-strips ? \(J \pi\) )
        using trace-parallel-plan-strips-last-cons-then nb
        by metis
        hence \(G \subseteq_{m}\) last (trace-parallel-plan-strips ? \(J \pi\) )
            using Cons.prems(2)
            by force
        \}
        moreover \{
            fix \(k\)
            assume \(k<\) length \(\pi\)
            moreover have \(k+1<\) length (ops \# \(\pi\) )
                using calculation
                by force
            moreover have \(\pi!k=(\) ops \(\# \pi)!(k+1)\)
                by \(\operatorname{simp}\)
            ultimately have are-all-operators-applicable
```

```
        (trace-parallel-plan-strips ?J \pi!k) ( }\pi!k
        and are-all-operator-effects-consistent ( }\pi!k
        and are-all-operators-non-interfering ( }\pi!k
        using Cons.prems(3) nb
        by force+
}
ultimately have G}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-serial-plan ?J (concat }\pi\mathrm{ )
    using Cons.IH[OF Cons.prems(1), of ?J]
    by blast
moreover {
    have execute-serial-plan I (concat (ops # \pi))
        = execute-serial-plan ? J' (concat \pi)
        using execute-serial-plan-split[of I ops] Cons.prems(3)
        by auto
    thm execute-parallel-operator-equals-execute-sequential-strips-if[of I]
    moreover have ?J = ? 'J'
    using execute-parallel-operator-equals-execute-sequential-strips-if Cons.prems(3)
        by fastforce
    ultimately have execute-serial-plan I (concat (ops # \pi)}\mathrm{ )
        = execute-serial-plan ?J (concat \pi)
        using execute-serial-plan-split[of I ops] Cons.prems(3)
        by argo
}
ultimately show ?case
    by argo
qed force
```


### 3.4 Auxiliary lemmas about STRIPS

lemma set-to-precondition-of-op-is[simp]: set (to-precondition op) $=\{(v$, True $) \mid v . v \in \operatorname{set}$ (precondition-of op) $\}$
unfolding to-precondition-def STRIPS-Representation.to-precondition-def set-map by blast
end
theory SAS-Plus-Representation
imports State-Variable-Representation
begin

## 4 SAS+ Representation

We now continue by defining a concrete implementation of SAS+.
SAS+ operators and SAS+ problems again use records. In contrast to STRIPS, the operator effect is contracted into a single list however since we now potentially deal with more than two possible values for each problem variable.
record ('variable, 'domain) sas-plus-operator $=$ precondition-of :: ('variable, 'domain) assignment list effect-of :: ('variable, 'domain) assignment list
record ('variable, 'domain) sas-plus-problem $=$
variables-of :: 'variable list ((-v+) [1000] 999)
operators-of :: ('variable, 'domain) sas-plus-operator list ((-О+) [1000] 999)
initial-of :: ('variable, 'domain) state ((-I+) [1000] 999)
goal-of :: ('variable, 'domain) state ((-G+) [1000] 999)
range-of :: 'variable - 'domain list
definition range-of':: ('variable, 'domain) sas-plus-problem $\Rightarrow$ 'variable $\Rightarrow$ 'domain set $\left(\mathcal{R}_{+}--52\right)$
where
range-of ${ }^{\prime} \Psi v \equiv$ (case sas-plus-problem.range-of $\Psi v$ of None $\Rightarrow\}$
| Some as $\Rightarrow$ set as)
definition to-precondition
:: ('variable, 'domain) sas-plus-operator $\Rightarrow$ ('variable, 'domain) assignment list where to-precondition $\equiv$ precondition-of
definition to-effect
:: ('variable, 'domain) sas-plus-operator $\Rightarrow$ ('variable, 'domain) Effect
where to-effect $o p \equiv[(v, a) .(v, a) \leftarrow$ effect-of $o p]$
type-synonym ('variable, 'domain) sas-plus-plan
$=($ 'variable, 'domain) sas-plus-operator list
type-synonym ('variable, 'domain) sas-plus-parallel-plan
$=($ 'variable, 'domain) sas-plus-operator list list
abbreviation empty-operator
:: ('variable, 'domain) sas-plus-operator ( $\varrho$ )
where empty-operator $\equiv$ ( precondition-of $=[]$, effect-of $=[]$ )
definition is-valid-operator-sas-plus
:: ('variable, 'domain) sas-plus-problem $\Rightarrow$ ('variable, 'domain) sas-plus-operator $\Rightarrow$ bool
where is-valid-operator-sas-plus $\Psi$ op $\equiv$ let
pre $=$ precondition - of op
; eff $=$ effect-of op
; vs $=$ variables-of $\Psi$
; $D=$ range-of $\Psi$
in list-all $(\lambda(v, a)$. ListMem $v$ vs) pre
$\wedge$ list-all $(\lambda(v, a) .(D v \neq$ None $) \wedge$ ListMem a $($ the $(D v)))$ pre
$\wedge$ list-all $(\lambda(v, a)$. ListMem $v$ vs) eff
$\wedge$ list-all $(\lambda(v, a) .(D v \neq$ None $) \wedge$ ListMem a $($ the $(D v)))$ eff
$\wedge$ list-all $\left(\lambda(v, a)\right.$. list-all $\left(\lambda\left(v^{\prime}, a^{\prime}\right) . v \neq v^{\prime} \vee a=a^{\prime}\right)$ pre) pre

$$
\wedge \text { list-all }\left(\lambda(v, a) . \text { list-all }\left(\lambda\left(v^{\prime}, a^{\prime}\right) . v \neq v^{\prime} \vee a=a^{\prime}\right) \text { eff }\right) \text { eff }
$$

definition is-valid-problem-sas-plus $\Psi$

$$
\equiv \text { let ops }=\text { operators-of } \Psi
$$

; vs $=$ variables-of $\Psi$
; $I=$ initial-of $\Psi$
; $G=$ goal-of $\Psi$
; $D=$ range-of $\Psi$
in list-all ( $\lambda v . D v \neq$ None) vs
$\wedge$ list-all (is-valid-operator-sas-plus $\Psi$ ) ops
$\wedge(\forall v . I v \neq$ None $\longleftrightarrow$ ListMem $v$ vs $)$
$\wedge(\forall v . I v \neq$ None $\longrightarrow$ ListMem $($ the $(I v))($ the $(D v)))$
$\wedge(\forall v . G v \neq$ None $\longrightarrow$ ListMem $v($ variables-of $\Psi))$
$\wedge(\forall v . G v \neq$ None $\longrightarrow$ ListMem $($ the $(G v))($ the $(D v)))$
definition is-operator-applicable-in
:: ('variable, 'domain) state
$\Rightarrow$ ('variable, 'domain) sas-plus-operator
$\Rightarrow$ bool
where is-operator-applicable-in s op
$\equiv$ map-of (precondition-of op) $\subseteq_{m} s$
definition execute-operator-sas-plus
:: ('variable, 'domain) state
$\Rightarrow$ ('variable, 'domain) sas-plus-operator
$\Rightarrow$ ('variable, 'domain) state (infixl $>_{+}$52)
where execute-operator-sas-plus s op $\equiv s++$ map-of (effect-of op)

- Set up simp rules to keep use of local parameters transparent within proofs (i.e. automatically substitute definitions).
lemma[simp]:
is-operator-applicable-in sop $=\left(\right.$ map-of $($ precondition-of op $\left.) \subseteq_{m} s\right)$
$s>_{+} o p=s++$ map-of (effect-of op)
unfolding initial-of-def goal-of-def variables-of-def range-of-def operators-of-def
SAS-Plus-Representation.is-operator-applicable-in-def
SAS-Plus-Representation.execute-operator-sas-plus-def
by $\operatorname{simp}+$
lemma range-of-not-empty:
(sas-plus-problem.range-of $\Psi v \neq$ None $\wedge$ sas-plus-problem.range-of $\Psi v \neq$ Some [])
$\longleftrightarrow\left(\mathcal{R}_{+} \Psi v\right) \neq\{ \}$
apply (cases sas-plus-problem.range-of $\Psi v$ )
by (auto simp add: SAS-Plus-Representation.range-of'-def)
lemma is-valid-operator-sas-plus-then:
fixes $\Psi::\left({ }^{\prime} v, ' d\right)$ sas-plus-problem

```
assumes is-valid-operator-sas-plus \(\Psi\) op
shows \(\forall(v, a) \in \operatorname{set}(\) precondition-of op \() . v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
    and \(\forall(v, a) \in \operatorname{set}\left(\right.\) precondition-of op). \(\left(\mathcal{R}_{+} \Psi v\right) \neq\{ \} \wedge a \in \mathcal{R}_{+} \Psi v\)
    and \(\forall(v, a) \in \operatorname{set}(\) effect-of op \() . v \in \operatorname{set}\left((\Psi)_{\mathcal{V}+}\right)\)
    and \(\forall(v, a) \in \operatorname{set}\left(\right.\) effect-of op). \(\left(\mathcal{R}_{+} \Psi v\right) \neq\{ \} \wedge a \in \mathcal{R}_{+} \Psi v\)
    and \(\forall(v, a) \in\) set (precondition-of op). \(\forall\left(v^{\prime}, a^{\prime}\right) \in\) set (precondition-of op). \(v\)
\(\neq v^{\prime} \vee a=a^{\prime}\)
    and \(\forall(v, a) \in \operatorname{set}(\) effect-of op).
        \(\forall\left(v^{\prime}, a^{\prime}\right) \in \operatorname{set}(\) effect-of op \() . v \neq v^{\prime} \vee a=a^{\prime}\)
proof -
    let ?vs \(=\) sas-plus-problem.variables-of \(\Psi\)
        and ?pre \(=\) precondition-of op
        and ?eff \(=\) effect-of op
        and \(? D=\) sas-plus-problem.range-of \(\Psi\)
    have \(\forall(v, a) \in\) set ? pre. \(v \in\) set ?vs
    and \(\forall(v, a) \in\) set ?pre.
        \((? D v \neq\) None \() \wedge\)
        \(a \in \operatorname{set}(\) the \((? D v))\)
    and \(\forall(v, a) \in\) set ?eff. \(v \in\) set ?vs
    and \(\forall(v, a) \in\) set ?eff.
        \((? D v \neq\) None \() \wedge\)
        \(a \in \operatorname{set}(\) the \((? D v))\)
    and \(\forall(v, a) \in\) set ?pre.
        \(\forall\left(v^{\prime}, a^{\prime}\right) \in\) set ? pre. \(v \neq v^{\prime} \vee a=a^{\prime}\)
    and \(\forall(v, a) \in\) set ?eff.
        \(\forall\left(v^{\prime}, a^{\prime}\right) \in\) set ?eff. \(v \neq v^{\prime} \vee a=a^{\prime}\)
    using assms
    unfolding is-valid-operator-sas-plus-def Let-def list-all-iff ListMem-iff
    by meson +
moreover have \(\forall(v, a) \in\) set ?pre. \(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
    and \(\forall(v, a) \in \operatorname{set}\) ?eff. \(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
    and \(\forall(v, a) \in\) set ?pre. \(\forall\left(v^{\prime}, a^{\prime}\right) \in\) set ?pre. \(v \neq v^{\prime} \vee a=a^{\prime}\)
    and \(\forall(v, a) \in\) set ?eff. \(\forall\left(v^{\prime}, a^{\prime}\right) \in\) set ?eff. \(v \neq v^{\prime} \vee a=a^{\prime}\)
    using calculation
    unfolding variables-of-def
    by blast+
moreover \{
    have \(\forall(v, a) \in\) set ?pre. \((? D v \neq\) None \() \wedge a \in \operatorname{set}(\) the \((? D v))\)
        using assms
        unfolding is-valid-operator-sas-plus-def Let-def list-all-iff ListMem-iff
        by argo
    hence \(\forall(v, a) \in\) set ?pre. \(\left(\left(\mathcal{R}_{+} \Psi v\right) \neq\{ \}\right) \wedge a \in \mathcal{R}_{+} \Psi v\)
        using range-of'-def
        by fastforce
\}
moreover \{
    have \(\forall(v, a) \in\) set ?eff. (?D \(v \neq\) None \() \wedge a \in \operatorname{set}(\) the \((? D v))\)
        using assms
        unfolding is-valid-operator-sas-plus-def Let-def list-all-iff ListMem-iff
```

```
        by argo
    hence }\forall(v,a)\in\mathrm{ set ?eff. ((}\mp@subsup{\mathcal{R}}{+}{}\Psiv)\not={})\wedgea\in\mp@subsup{\mathcal{R}}{+}{}\Psi
        using range-of'-def
        by fastforce
    }
    ultimately show }\forall(v,a)\in\operatorname{set}(precondition-of op). v\in set ((\Psi)\mp@subsup{\mathcal{V}}{+}{}
    and}\forall(v,a)\in\operatorname{set (precondition-of op). (\mathcal{R}}\mp@subsup{\mathcal{+}}{+}{\Psi}v)\not={}\wedgea\in\mp@subsup{\mathcal{R}}{+}{}\Psi
    and}\forall(v,a)\in\operatorname{set}(effect-of op).v\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{V}+}{}
    and}\forall(v,a)\in\operatorname{set}(effect-of op). (\mathcal{R}+\Psiv)\not={}\wedgea\in\mathcal{R}+\Psi
    and }\forall(v,a)\in\mathrm{ set (precondition-of op).}\forall(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\mathrm{ set (precondition-of op).v
# v
    and}\forall(v,a)\in\operatorname{set}(effect-of op)
        \forall(\mp@subsup{v}{}{\prime},a},\mp@subsup{a}{}{\prime})\in\operatorname{set}(\mathrm{ effect-of op). v}\not=\mp@subsup{v}{}{\prime}\veea=\mp@subsup{a}{}{\prime
    by blast+
qed
```


## lemma is-valid-problem-sas-plus-then:

fixes $\Psi::\left({ }^{\prime} v, ' d\right)$ sas-plus-problem
assumes is-valid-problem-sas-plus $\Psi$
shows $\forall v \in \operatorname{set}\left((\Psi)_{\mathcal{V}+}\right) .\left(\mathcal{R}_{+} \Psi v\right) \neq\{ \}$
and $\forall o p \in$ set $\left((\Psi)_{\mathcal{O}_{+}}\right)$. is-valid-operator-sas-plus $\Psi$ op
and $\operatorname{dom}\left((\Psi)_{I+}\right)=\operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)$
and $\forall v \in \operatorname{dom}\left((\Psi)_{I+}\right)$. the $\left(\left((\Psi)_{I+}\right) v\right) \in \mathcal{R}_{+} \Psi v$
and $\operatorname{dom}\left((\Psi)_{G+}\right) \subseteq \operatorname{set}\left((\Psi)_{\mathcal{V}}\right)$
and $\forall v \in \operatorname{dom}\left((\Psi)_{G+}\right)$. the $\left(\left((\Psi)_{G+}\right) v\right) \in \mathcal{R}_{+} \Psi v$
proof -
let ?vs $=$ sas-plus-problem.variables-of $\Psi$
and $?$ ops $=$ sas-plus-problem.operators-of $\Psi$
and $? I=$ sas-plus-problem.initial-of $\Psi$
and $? G=$ sas-plus-problem.goal-of $\Psi$
and $? D=$ sas-plus-problem.range-of $\Psi$
\{
fix $v$
have $(? D \quad v \neq$ None $\wedge ? D v \neq$ Some []$) \longleftrightarrow\left(\left(\mathcal{R}_{+} \Psi v\right) \neq\{ \}\right)$
by (cases ?D $v$; (auto simp: range-of ${ }^{\prime}$-def))
$\}$ note $n b=$ this
have $n b_{1}: \forall v \in$ set ?vs. ?D $v \neq$ None
and $\forall o p \in$ set ?ops. is-valid-operator-sas-plus $\Psi$ op
and $\forall v$. $(? I v \neq$ None $)=(v \in$ set ? vs $)$
and $n b_{2}: \forall v$. ?I $v \neq$ None $\longrightarrow$ the $($ ?I $v) \in$ set $($ the $(? D v))$
and $\forall v$. ? $G v \neq$ None $\longrightarrow v \in$ set ? $v s$
and $n b_{3}: \forall v$. ?G $v \neq$ None $\longrightarrow$ the $(? G v) \in \operatorname{set}($ the $(? D v))$
using assms
unfolding SAS-Plus-Representation.is-valid-problem-sas-plus-def Let-def list-all-iff ListMem-iff
by argo+
then have G3: $\forall o p \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)$. is-valid-operator-sas-plus $\Psi$ op
and $G 4: \operatorname{dom}\left((\Psi)_{I+}\right)=\operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)$

```
    and G5:dom ((\Psi)}\mp@subsup{)}{G+}{})\subseteq\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{V}+}{}
    unfolding variables-of-def operators-of-def
    by auto+
moreover {
    fix v
    assume v\in set ((\Psi) (\mathcal{V}+)
    then have ?D v\not= None
        using nb
    by force+
} note G6 = this
moreover {
    fix v
    assume v\in\operatorname{dom}((\Psi\mp@subsup{)}{I+}{})
    moreover have ((\Psi\mp@subsup{)}{I+}{\prime})v\not= None
        using calculation
        by blast+
    moreover {
        have v\in set ((\Psi) (\mathcal{V}+
            using G4 calculation(1)
            by argo
        then have sas-plus-problem.range-of \Psi v}\not=\mathrm{ None
            using range-of-not-empty
            unfolding range-of'-def
            using G6
            by fast+
```



```
            by (simp add: <sas-plus-problem.range-of \Psi v = None〉 option.case-eq-if
range-of'-def)
    }
    ultimately have the }(((\Psi\mp@subsup{)}{I+}{})v)\in\mp@subsup{\mathcal{R}}{+}{}\Psi
        using n\mp@subsup{b}{2}{}
        by force
}
moreover {
    fix v
    assume v\in dom ((\Psi) G+)
    then have ((\Psi)}\mp@subsup{)}{G+}{})v\not= Non
        by blast
    moreover {
        have v\in set ((\Psi) (\mathcal{V}+
            using G5 calculation(1)
            by fast
            then have sas-plus-problem.range-of \Psi v}\not=\mathrm{ None
            using range-of-not-empty
            using G6
            by fast+
```



```
            by (simp add:<sas-plus-problem.range-of \Psi v = None〉 option.case-eq-if
range-of'-def)
```

```
    }
    ultimately have the (((\Psi)}\mp@subsup{)}{G+}{})v)\in\mp@subsup{\mathcal{R}}{+}{}\Psi
        using nb 3
        by auto
    }
    ultimately show }\forallv\in\operatorname{set}((\Psi\mp@subsup{)}{\mp@subsup{\mathcal{V}}{+}{\prime}}{}).(\mp@subsup{\mathcal{R}}{+}{}\Psiv)\not={
    and }\forallop\in\operatorname{set}((\Psi\mp@subsup{)}{\mp@subsup{\mathcal{O}}{+}{\prime}}{)}\mathrm{ . is-valid-operator-sas-plus }\Psi\mathrm{ op
    and dom ((\Psi) I+ ) = set ((\Psi) V+ )
    and }\forallv\in\operatorname{dom}((\Psi\mp@subsup{)}{I+}{\prime})\mathrm{ . the }(((\Psi\mp@subsup{)}{I+}{})v)\in\mp@subsup{\mathcal{R}}{+}{}\Psi
    and dom ((\Psi) (G+)\subseteq set ((\Psi)
    and }\forallv\in\operatorname{dom}((\Psi\mp@subsup{)}{G+}{})\mathrm{ . the }(((\Psi\mp@subsup{)}{G+}{})v)\in\mp@subsup{\mathcal{R}}{+}{}\Psi
    by blast+
qed
end
theory SAS-Plus-Semantics
    imports SAS-Plus-Representation List-Supplement
    Map-Supplement
begin
```


## 5 SAS+ Semantics

### 5.1 Serial Execution Semantics

Serial plan execution is implemented recursively just like in the STRIPS case. By and large, compared to definition ??, we only substitute the operator applicability function with its SAS+ counterpart.

```
primrec execute-serial-plan-sas-plus
    where execute-serial-plan-sas-plus s[] =s
    | execute-serial-plan-sas-plus s (op # ops)
        = (if is-operator-applicable-in sop
    then execute-serial-plan-sas-plus (execute-operator-sas-plus s op) ops
    else s)
```

Similarly, serial SAS+ solutions are defined just like in STRIPS but based on the corresponding SAS+ definitions.

```
definition is-serial-solution-for-problem
    :: ('variable, 'domain) sas-plus-problem \(\Rightarrow\) ('variable, 'domain) sas-plus-plan \(\Rightarrow\)
bool
    where is-serial-solution-for-problem \(\Psi \psi\)
        三 let
            \(I=\) sas-plus-problem.initial-of \(\Psi\)
            ; \(G=\) sas-plus-problem.goal-of \(\Psi\)
            ; ops \(=\) sas-plus-problem.operators-of \(\Psi\)
            in \(G \subseteq_{m}\) execute-serial-plan-sas-plus \(I \psi\)
            \(\wedge\) list-all ( (op. ListMem op ops) \(\psi\)
```

```
context
begin
private lemma execute-operator-sas-plus-effect-i:
    assumes is-operator-applicable-in s op
        and }\forall(v,a)\in\operatorname{set}(effect-of op).\forall(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\operatorname{set}(effect-of op)
            v\not=\mp@subsup{v}{}{\prime}\veea=\mp@subsup{a}{}{\prime}
        and(v,a)\inset (effect-of op)
    shows (s>>+op) v=Some a
proof -
    let ?effect = effect-of op
    have map-of ?effect v=Some a
        using map-of-constant-assignments-defined-if[OF assms(2, 3)] try0
        by blast
    thus ?thesis
        unfolding execute-operator-sas-plus-def map-add-def
        by fastforce
qed
private lemma execute-operator-sas-plus-effect-ii:
    assumes is-operator-applicable-in s op
        and}\forall(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\operatorname{set}(effect-of op). v'\not=
    shows (s>>+op)v=sv
proof -
    let ?effect = effect-of op
    {
        have v\not\in fst ' set ?effect
            using assms(2)
            by fastforce
        then have v # dom (map-of ?effect)
            using dom-map-of-conv-image-fst[of ?effect]
            by argo
        hence (s++ map-of ?effect) v=sv
            using map-add-dom-app-simps(3)[of v map-of ?effect s]
            by blast
    }
    thus ?thesis
        by fastforce
qed
```

Given an operator $o p$ that is applicable in a state $s$ and has a consistent set of effects (second assumption) we can now show that the successor state $s^{\prime}$ $\equiv s>_{+} o p$ has the following properties:

- $s^{\prime} v=$ Some $a$ if ( $v, a$ ) exist in set (effect-of op); and,
- $s^{\prime} v=s v$ if no $\left(v, a^{\prime}\right)$ exist in set (effect-of op).

The second property is the case if the operator doesn't have an effect for a
variable $v$.
theorem execute-operator-sas-plus-effect:
assumes is-operator-applicable-in s op
and $\forall(v, a) \in$ set (effect-of op). $\forall\left(v^{\prime}, a^{\prime}\right) \in \operatorname{set}\left(\right.$ effect-of op). $v \neq v^{\prime} \vee a=a^{\prime}$
shows $(v, a) \in$ set (effect-of op) $\longrightarrow(s \gg+o p) v=$ Some $a$
and $(\forall a .(v, a) \notin$ set (effect-of op))
$\longrightarrow\left(s>_{+} o p\right) v=s v$
proof -
show $(v, a) \in \operatorname{set}($ effect-of op)
$\longrightarrow(s \gg+o p) v=$ Some $a$
using execute-operator-sas-plus-effect-i[OF $\operatorname{assms}(1,2)]$
by blast
next
show $(\forall a .(v, a) \notin \operatorname{set}($ effect-of op))
$\longrightarrow\left(s>_{+} o p\right) v=s v$
using execute-operator-sas-plus-effect-ii[OF assms(1)]
by blast
qed
end

### 5.2 Parallel Execution Semantics

type-synonym ('variable, 'domain) sas-plus-parallel-plan $=($ 'variable, 'domain) sas-plus-operator list list
definition are-all-operators-applicable-in
:: ('variable, 'domain) state
$\Rightarrow$ ('variable, 'domain) sas-plus-operator list
$\Rightarrow$ bool
where are-all-operators-applicable-in s ops
$\equiv$ list-all (is-operator-applicable-in s) ops
definition are-operator-effects-consistent
:: ('variable, 'domain) sas-plus-operator
$\Rightarrow$ ('variable, 'domain) sas-plus-operator
$\Rightarrow$ bool
where are-operator-effects-consistent op op ${ }^{\prime}$
$\equiv$ let
effect $=$ effect-of op
; effect ${ }^{\prime}=$ effect-of op ${ }^{\prime}$
in list-all $\left(\lambda(v, a)\right.$. list-all $\left(\lambda\left(v^{\prime}, a^{\prime}\right) . v \neq v^{\prime} \vee a=a^{\prime}\right)$ effect') effect
definition are-all-operator-effects-consistent
:: ('variable, 'domain) sas-plus-operator list
$\Rightarrow$ bool
where are-all-operator-effects-consistent ops

$$
\equiv \text { list-all ( } \lambda o p . \text { list-all (are-operator-effects-consistent op) ops) ops }
$$

definition execute-parallel-operator-sas-plus
:: ('variable, 'domain) state
$\Rightarrow$ ('variable, 'domain) sas-plus-operator list
$\Rightarrow$ ('variable, 'domain) state
where execute-parallel-operator-sas-plus s ops
$\equiv$ foldl $(++) s($ map (map-of $\circ$ effect-of) ops)
We now define parallel execution and parallel traces for SAS+ by lifting the tests for applicability and effect consistency to parallel SAS+ operators. The definitions are again very similar to their STRIPS analogs (definitions ?? and ??).

```
fun execute-parallel-plan-sas-plus
    :: ('variable, 'domain) state
    # ('variable, 'domain) sas-plus-parallel-plan
    # ('variable, 'domain) state
    where execute-parallel-plan-sas-plus s[] =s
    | execute-parallel-plan-sas-plus s (ops # opss)=(if
        are-all-operators-applicable-in s ops
        \wedge ~ a r e - a l l - o p e r a t o r - e f f e c t s - c o n s i s t e n t ~ o p s
        then execute-parallel-plan-sas-plus
            (execute-parallel-operator-sas-plus s ops) opss
        else s)
fun trace-parallel-plan-sas-plus
    :: ('variable, 'domain) state
        =>('variable,'domain) sas-plus-parallel-plan
        => ('variable, 'domain) state list
    where trace-parallel-plan-sas-plus s [] = [s]
    | trace-parallel-plan-sas-plus s (ops # opss)}=s# (i
        are-all-operators-applicable-in s ops
        \wedge are-all-operator-effects-consistent ops
    then trace-parallel-plan-sas-plus
        (execute-parallel-operator-sas-plus s ops) opss
        else [])
```

A plan $\psi$ is a solution for a SAS+ problem $\Psi$ if

1. starting from the initial state $\Psi$, SAS+ parallel plan execution reaches a state which satisfies the described goal state $\Psi_{G+}$; and,
2. all parallel operators ops in the plan $\psi$ only consist of operators that are specified in the problem description.
definition is-parallel-solution-for-problem
:: ('variable, 'domain) sas-plus-problem
$\Rightarrow$ ('variable, 'domain) sas-plus-parallel-plan
$\Rightarrow$ bool
```
where is-parallel-solution-for-problem \(\Psi \psi\)
    \(\equiv\) let
    \(G=\) sas-plus-problem.goal-of \(\Psi\)
    ; \(I=\) sas-plus-problem.initial-of \(\Psi\)
    ; Ops \(=\) sas-plus-problem.operators-of \(\Psi\)
    in \(G \subseteq_{m}\) execute-parallel-plan-sas-plus I \(\psi\)
    \(\wedge\) list-all ( \(\lambda\) ops. list-all ( \(\lambda o p\). ListMem op Ops) ops) \(\psi\)
context
begin
lemma execute-parallel-operator-sas-plus-cons[simp]:
    execute-parallel-operator-sas-plus s (op \# ops)
    \(=\) execute-parallel-operator-sas-plus \((s++\) map-of (effect-of op)) ops
    unfolding execute-parallel-operator-sas-plus-def
    by \(\operatorname{simp}\)
```

The following lemmas show the properties of SAS+ parallel plan execution traces. The results are analogous to those for STRIPS. So, let $\tau \equiv$ trace-parallel-plan-sas-plus $I \psi$ be a trace of a parallel SAS+ plan $\psi$ with initial state $I$, then

- the head of the trace $\tau!0$ is the initial state of the problem (lemma ??); moreover,
- for all but the last element of the trace - i.e. elements with index $k<$ length $\tau-1$-the parallel operator $\pi!k$ is executable (lemma ??); and finally,
- for all $k<$ length $\tau$, the parallel execution of the plan prefix take $k \psi$ with initial state $I$ equals the $k$-th element of the trace $\tau!k$ (lemma ??).

```
lemma trace-parallel-plan-sas-plus-head-is-initial-state:
    trace-parallel-plan-sas-plus \(I \psi!0=I\)
proof (cases \(\psi\) )
    case (Cons a list)
    then show ?thesis
        by (cases are-all-operators-applicable-in I a \(\wedge\) are-all-operator-effects-consistent
\(a\);
        simp+)
qed \(\operatorname{simp}\)
lemma trace-parallel-plan-sas-plus-length-gt-one-if:
    assumes \(k<\) length (trace-parallel-plan-sas-plus I \(\psi\) ) - 1
    shows \(1<\) length (trace-parallel-plan-sas-plus I \(\psi\) )
    using assms
    by linarith
```

```
lemma length-trace-parallel-plan-sas-plus-lte-length-plan-plus-one:
    shows length (trace-parallel-plan-sas-plus \(I \psi\) ) \(\leq\) length \(\psi+1\)
proof (induction \(\psi\) arbitrary: I)
    case (Cons a \(\psi\) )
    then show? case
    proof (cases are-all-operators-applicable-in I a \(\wedge\) are-all-operator-effects-consistent
a)
    case True
    let \(? I^{\prime}=\) execute-parallel-operator-sas-plus \(I\) a
    \{
    have trace-parallel-plan-sas-plus \(I(a \# \psi)=I \#\) trace-parallel-plan-sas-plus
? \(I^{\prime} \psi\)
            using True
            by auto
            then have length (trace-parallel-plan-sas-plus \(I(a \# \psi))\)
                \(=\) length (trace-parallel-plan-sas-plus ? \(\left.I^{\prime} \psi\right)+1\)
                by \(\operatorname{simp}\)
            moreover have length (trace-parallel-plan-sas-plus ? \(\left.I^{\prime} \psi\right) \leq\) length \(\psi+1\)
            using Cons.IH[of ?I \(]\)
            by blast
            ultimately have length (trace-parallel-plan-sas-plus \(I(a \# \psi)) \leq l e n g t h ~(a\)
\(\# \psi)+1\)
            by \(\operatorname{simp}\)
    \}
        thus ?thesis
            by blast
    qed auto
qed simp
lemma plan-is-at-least-singleton-plan-if-trace-has-at-least-two-elements:
    assumes \(k<\) length (trace-parallel-plan-sas-plus I \(\psi\) ) - 1
    obtains ops \(\psi^{\prime}\) where \(\psi=\) ops \(\# \psi^{\prime}\)
proof -
    let ? \(\tau=\) trace-parallel-plan-sas-plus I \(\psi\)
    have length ? \(\tau \leq\) length \(\psi+1\)
        using length-trace-parallel-plan-sas-plus-lte-length-plan-plus-one
        by fast
    then have \(0<\) length \(\psi\)
    using trace-parallel-plan-sas-plus-length-gt-one-if[OF assms]
    by fastforce
    then obtain \(k^{\prime}\) where length \(\psi=\) Suc \(k^{\prime}\)
        using gr0-implies-Suc
        by meson
    thus ?thesis using that
    using length-Suc-conv[of \(\left.\psi k^{\prime}\right]\)
    by blast
qed
```

lemma trace-parallel-plan-sas-plus-step-implies-operator-execution-condition-holds:

```
    assumes \(k<\) length (trace-parallel-plan-sas-plus \(I \pi\) ) - 1
```

    shows are-all-operators-applicable-in (trace-parallel-plan-sas-plus \(I \pi!k)(\pi!k)\)
    \(\wedge\) are-all-operator-effects-consistent \((\pi!k)\)
    using assms
proof (induction $\pi$ arbitrary: I k)

- NOTE Base case yields contradiction with assumption and can be left to automation.
case (Cons a $\pi$ )
then show? case
proof (cases are-all-operators-applicable-in I a $\wedge$ are-all-operator-effects-consistent
a)

```
case True
have trace-parallel-plan-sas-plus-cons: trace-parallel-plan-sas-plus I (a# #)
    = I # trace-parallel-plan-sas-plus (execute-parallel-operator-sas-plus I a) \pi
    using True
    by simp
    then show ?thesis
    proof (cases k)
    case 0
    have trace-parallel-plan-sas-plus I (a# \pi)!0 = I
        using trace-parallel-plan-sas-plus-cons
        by simp
    moreover have (a#\pi)!0=a
        by simp
    ultimately show ?thesis
        using True 0
        by presburger
    next
    case (Suc k')
    have trace-parallel-plan-sas-plus I (a#\pi)!Suc k'
        = trace-parallel-plan-sas-plus (execute-parallel-operator-sas-plus I a) \pi! k'
        using trace-parallel-plan-sas-plus-cons
        by simp
```



```
        by simp
    moreover {
        let ?I' = execute-parallel-operator-sas-plus I a
        have length (trace-parallel-plan-sas-plus I (a # \pi))
        =1 + length (trace-parallel-plan-sas-plus ?I' }\pi\mathrm{ )
        using trace-parallel-plan-sas-plus-cons
        by auto
    then have }\mp@subsup{k}{}{\prime}<length (trace-parallel-plan-sas-plus ?I' \pi) - 1
                using Cons.prems Suc
                unfolding Suc-eq-plus1
        by fastforce
        hence are-all-operators-applicable-in
        (trace-parallel-plan-sas-plus (execute-parallel-operator-sas-plus I a) \pi! k')
                (\pi!k}
        \wedge are-all-operator-effects-consistent ( }\pi!\mp@subsup{k}{}{\prime}
```

using Cons.IH[of $k^{\prime}$ execute-parallel-operator-sas-plus I a] Cons.prems Suc trace-parallel-plan-sas-plus-cons
by $\operatorname{simp}$
\}
ultimately show ?thesis
using Suc
by argo
qed
next
case False
then have trace-parallel-plan-sas-plus $I(a \# \pi)=[I]$
by force
then have length (trace-parallel-plan-sas-plus $I(a \# \pi))-1=0$ by simp

- NOTE Thesis follows from contradiction with assumption.
then show ?thesis
using Cons.prems
by force
qed
qed auto
lemma trace-parallel-plan-sas-plus-prefix:
assumes $k<$ length (trace-parallel-plan-sas-plus I $\psi$ )
shows trace-parallel-plan-sas-plus $I \psi!k=$ execute-parallel-plan-sas-plus I (take $k \psi$ )
using assms
proof (induction $\psi$ arbitrary: I k)
case (Cons a $\psi$ )
then show ?case
proof (cases are-all-operators-applicable-in I a $\wedge$ are-all-operator-effects-consistent
a)
case True
let $? \sigma=$ trace-parallel-plan-sas-plus $I(a \# \psi)$
and $? I^{\prime}=$ execute-parallel-operator-sas-plus I a
have $\sigma$-equals: ? $\sigma=I \#$ trace-parallel-plan-sas-plus ? $I^{\prime} \psi$
using True
by auto
then show ?thesis
proof (cases $k=0$ )
case False
obtain $k^{\prime}$ where $k$-is-suc-of- $k^{\prime}: k=$ Suc $k^{\prime}$
using not0-implies-Suc[OF False]
by blast
then have execute-parallel-plan-sas-plus I (take $k(a \# \psi))$
$=$ execute-parallel-plan-sas-plus ? $I^{\prime}\left(\right.$ take $\left.k^{\prime} \psi\right)$
using True
by $\operatorname{simp}$
moreover have trace-parallel-plan-sas-plus $I(a \# \psi)!k$
$=$ trace-parallel-plan-sas-plus ? $I^{\prime} \psi!k^{\prime}$

```
            using \sigma-equals k-is-suc-of-k'
            by simp
            moreover {
                have }\mp@subsup{k}{}{\prime}<length (trace-parallel-plan-sas-plus ?I' \psi
                using Cons.prems }\sigma\mathrm{ -equals }k\mathrm{ -is-suc-of-k'
                by force
            hence trace-parallel-plan-sas-plus ?I' }\psi!\mp@subsup{k}{}{\prime
                = execute-parallel-plan-sas-plus ?I' (take k' \psi)
                using Cons.IH[of k' ?I']
                by blast
            }
            ultimately show ?thesis
            by presburger
        qed simp
    next
        case operator-precondition-violated: False
        then show ?thesis
    proof (cases k=0)
        case False
        then have trace-parallel-plan-sas-plus I (a#\psi)=[I]
            using operator-precondition-violated
            by force
    moreover have execute-parallel-plan-sas-plus I (take k (a#\psi))=I
                using Cons.prems operator-precondition-violated
                by force
            ultimately show ?thesis
                using Cons.prems nth-Cons-0
                by auto
    qed simp
    qed
qed simp
lemma trace-parallel-plan-sas-plus-step-effect-is:
    assumes }k<length (trace-parallel-plan-sas-plus I \psi) - 1
    shows trace-parallel-plan-sas-plus I }\psi!\mathrm{ Suc k
    = execute-parallel-operator-sas-plus (trace-parallel-plan-sas-plus I \psi!k)(\psi!k)
proof -
    let ?\tau = trace-parallel-plan-sas-plus I \psi
    let ? }\mp@subsup{\tau}{k}{}=?\tau\mp@code{\
        and ?}\mp@subsup{\tau}{k}{\prime}\mp@subsup{}{}{\prime}=?\tau\tau!\mathrm{ Suc k
    - NOTE rewrite the goal using the subplan formulation to be able. This allows
us to make the initial state arbitrary.
    {
        have suc-k-lt-length-\tau:Suc k< length ?\tau
            using assms
            by linarith
        hence ? ? }\mp@subsup{\tau}{k}{\prime}== execute-parallel-plan-sas-plus I (take (Suc k)\psi
            using trace-parallel-plan-sas-plus-prefix[of Suc k]
```

by blast
$\}$ note rewrite-goal $=$ this
have execute-parallel-plan-sas-plus I (take (Suc k) $\psi$ )
$=$ execute-parallel-operator-sas-plus (trace-parallel-plan-sas-plus $I \psi!k)(\psi!k)$
using assms
proof (induction $k$ arbitrary: $I \psi$ )
case 0
obtain ops $\psi^{\prime}$ where $\psi$-is: $\psi=$ ops $\# \psi^{\prime}$
using plan-is-at-least-singleton-plan-if-trace-has-at-least-two-elements[OF
0.prems]
by force
\{
have take (Suc 0) $\psi=\left[\begin{array}{l}\psi!0]\end{array}\right.$
using $\psi$-is
by $\operatorname{simp}$
hence execute-parallel-plan-sas-plus I (take (Suc 0) $\psi$ )
$=$ execute-parallel-plan-sas-plus $I[\psi!0]$
by argo
\}
moreover \{
have trace-parallel-plan-sas-plus $I \psi!0=I$
using trace-parallel-plan-sas-plus-head-is-initial-state.
moreover \{
have are-all-operators-applicable-in $I(\psi!0)$
and are-all-operator-effects-consistent $(\psi!0)$
using trace-parallel-plan-sas-plus-step-implies-operator-execution-condition-holds[OF O.prems] calculation
by argo+
then have execute-parallel-plan-sas-plus $I[\psi!0]$
$=$ execute-parallel-operator-sas-plus $I(\psi!0)$
by $\operatorname{simp}$
\}
ultimately have execute-parallel-operator-sas-plus (trace-parallel-plan-sas-plus $I \psi!0)$
$(\psi!0)$
$=$ execute-parallel-plan-sas-plus $I[\psi!0]$
by argo
\}
ultimately show ?case
by argo
next
case (Suc k)
obtain ops $\psi^{\prime}$ where $\psi-i s: \psi=o p s \# \psi^{\prime}$
using plan-is-at-least-singleton-plan-if-trace-has-at-least-two-elements[OF
Suc.prems]
by blast
let $? I^{\prime}=$ execute-parallel-operator-sas-plus I ops
have execute-parallel-plan-sas-plus I (take (Suc (Suc k)) $\psi$ )

```
            = execute-parallel-plan-sas-plus ?I' (take (Suc k) \psi')
            using Suc.prems \psi-is
            by fastforce
    moreover {
            thm Suc.IH[of ]
            have length (trace-parallel-plan-sas-plus I \psi)
                =1 + length (trace-parallel-plan-sas-plus ?I' \psi')
                using \psi-is Suc.prems
                by fastforce
            moreover have k< length (trace-parallel-plan-sas-plus ?I' \psi') - 1
            using Suc.prems calculation
            by fastforce
            ultimately have execute-parallel-plan-sas-plus ?I' (take (Suc k) \psi')=
                execute-parallel-operator-sas-plus (trace-parallel-plan-sas-plus ?I' \psi'!k)
                ( }\mp@subsup{\psi}{}{\prime}!k
                using Suc.IH[of ?I' \psi']
            by blast
    }
    moreover have execute-parallel-operator-sas-plus (trace-parallel-plan-sas-plus
? I' }\mp@subsup{\psi}{}{\prime}!k
            (\psi'! ! k)
    = execute-parallel-operator-sas-plus (trace-parallel-plan-sas-plus I \psi!Suc k)
                (\psi!Suc k)
            using Suc.prems \psi-is
            by auto
            ultimately show ?case
                by argo
    qed
thus ?thesis
    using rewrite-goal
    by argo
qed
```

Finally, we obtain the result corresponding to lemma ?? in the SAS+ case: it is equivalent to say that parallel SAS+ execution reaches the problem's goal state and that the last element of the corresponding trace satisfies the goal state.
lemma execute-parallel-plan-sas-plus-reaches-goal-iff-goal-is-last-element-of-trace:
$G \subseteq_{m}$ execute-parallel-plan-sas-plus I $\psi$
$\longleftrightarrow G \subseteq_{m}$ last (trace-parallel-plan-sas-plus I $\psi$ )
proof -
let ? $\tau=$ trace-parallel-plan-sas-plus $I \psi$
show ?thesis
proof (rule iffI)
assume $G \subseteq_{m}$ execute-parallel-plan-sas-plus $I \psi$
thus $G \subseteq_{m}$ last ? $\tau$
proof (induction $\psi$ arbitrary: I)

- NOTE Base case follows from simplification.
case (Cons ops $\psi$ )

```
        show ?case
    proof (cases are-all-operators-applicable-in I ops
        \ are-all-operator-effects-consistent ops)
        case True
        let ?s = execute-parallel-operator-sas-plus I ops
        {
            have G\subseteq\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan-sas-plus ?s }\psi=\mp@code{}|
                using True Cons.prems
                by simp
            hence G}\mp@subsup{\subseteq}{m}{}\mathrm{ last (trace-parallel-plan-sas-plus ?s }\psi\mathrm{ )
                using Cons.IH
            by auto
        }
        moreover {
            have trace-parallel-plan-sas-plus I (ops # \psi)
                = I # trace-parallel-plan-sas-plus ?s \psi
                using True
                by simp
            moreover have trace-parallel-plan-sas-plus ?s \psi \not=[]
                using trace-parallel-plan-sas-plus.elims
                by blast
            ultimately have last (trace-parallel-plan-sas-plus I (ops#\psi))
                = last (trace-parallel-plan-sas-plus ?s \psi)
                using last-ConsR
                by simp
        }
        ultimately show ?thesis
            by argo
    next
        case False
        then have G\subseteqm}
            using Cons.prems
            by force
            thus ?thesis
            using False
            by force
    qed
qed force
next
    assume G \subseteqm last ?\tau
    thus }G\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan-sas-plus I }
    proof (induction \psi arbitrary:I)
        case (Cons ops \psi)
        thus ?case
            proof (cases are-all-operators-applicable-in I ops
            \wedge ~ a r e - a l l - o p e r a t o r - e f f e c t s - c o n s i s t e n t ~ o p s )
            case True
            let ?s = execute-parallel-operator-sas-plus I ops
            {
```

```
                    have trace-parallel-plan-sas-plus I (ops # \psi)
                    = I # trace-parallel-plan-sas-plus ?s \psi
                    using True
                    by simp
                    moreover have trace-parallel-plan-sas-plus ?s \psi = []
                    using trace-parallel-plan-sas-plus.elims
                    by blast
            ultimately have last (trace-parallel-plan-sas-plus I (ops # \psi))
                    = last (trace-parallel-plan-sas-plus ?s \psi)
                    using last-ConsR
                    by simp
                    hence G}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan-sas-plus ?s }
                    using Cons.IH[of ?s] Cons.prems
                    by argo
            }
            moreover have execute-parallel-plan-sas-plus I (ops # \psi)
                    = execute-parallel-plan-sas-plus ?s \psi
                    using True
                    by force
            ultimately show ?thesis
                    by argo
            next
                    case False
                    have G}\mp@subsup{\subseteq}{m}{}
                            using Cons.prems False
                    by simp
                    thus?thesis
                    using False
                    by force
            qed
        qed simp
    qed
qed
lemma is-parallel-solution-for-problem-plan-operator-set:
fixes \(\Psi::(' v, ' d)\) sas-plus-problem
assumes is-parallel-solution-for-problem \Psi \psi
shows }\forall\mathrm{ ops }\in\mathrm{ set }\psi.\forallop\in set ops. op \in set ((\Psi) (%) 
using assms
unfolding is-parallel-solution-for-problem-def list-all-iff ListMem-iff operators-of-def
    by presburger
end
```


### 5.3 Serializable Parallel Plans

Again we want to establish conditions for the serializability of plans. Let $\Psi$ be a SAS+ problem instance and let $\psi$ be a serial solution. We obtain the following two important results, namely that

1. the embedding List-Supplement.embed $\psi$ of $\psi$ is a parallel solution for $\Psi$ (lemma ??); and conversely that,
2. a parallel solution to $\Psi$ that has the form of an embedded serial plan can be concatenated to obtain a serial solution (lemma ??).

## context

begin
lemma execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus-i:
assumes is-operator-applicable-in s op
are-operator-effects-consistent op op
shows $s \gg+o p=$ execute-parallel-operator-sas-plus $s[o p]$
proof -
have are-all-operators-applicable-in s [op]
unfolding are-all-operators-applicable-in-def
SAS-Plus-Representation.execute-operator-sas-plus-def
is-operator-applicable-in-def SAS-Plus-Representation.is-operator-applicable-in-def
list-all-iff
using assms(1)
by fastforce
moreover have are-all-operator-effects-consistent [op]
unfolding are-all-operator-effects-consistent-def list-all-iff using $\operatorname{assms}(2)$
by fastforce
ultimately show ?thesis
unfolding execute-parallel-operator-sas-plus-def execute-operator-sas-plus-def
by $\operatorname{simp}$
qed
lemma execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus-ii:
fixes $I::$ ('variable, 'domain) state
assumes $\forall o p \in$ set $\psi$. are-operator-effects-consistent op op
and $G \subseteq_{m}$ execute-serial-plan-sas-plus $I \psi$
shows $G \subseteq_{m}$ execute-parallel-plan-sas-plus I (embed $\psi$ )
using assms
proof (induction $\psi$ arbitrary: I)
case (Cons op $\psi$ )
show ?case
proof (cases are-all-operators-applicable-in $I[o p]$ )
case True
let ? $J=$ execute-operator-sas-plus I op

```
    let ?J' = execute-parallel-operator-sas-plus I [op]
    have SAS-Plus-Representation.is-operator-applicable-in I op
        using True
        unfolding are-all-operators-applicable-in-def list-all-iff
        by force
    moreover have G\subseteq}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-serial-plan-sas-plus ?J }
        using Cons.prems(2) calculation(1)
        by simp
    moreover have are-all-operator-effects-consistent [op]
        unfolding are-all-operator-effects-consistent-def list-all-iff Let-def
        using Cons.prems(1)
        by simp
    moreover have execute-parallel-plan-sas-plus I ([op] # embed \psi)
        = execute-parallel-plan-sas-plus ?J'}(\mathrm{ embed }\psi
        using True calculation(3)
        by simp
    moreover {
        have is-operator-applicable-in I op
            are-operator-effects-consistent op op
            using True Cons.prems(1)
            unfolding are-all-operators-applicable-in-def
                SAS-Plus-Representation.is-operator-applicable-in-def list-all-iff
            by fastforce+
    hence ?J = ? J'
            using execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus-i
                calculation(1)
            by blast
    }
    ultimately show ?thesis
        using Cons.IH[of ?J] Cons.prems(1)
        by simp
    next
    case False
    moreover have }\neg\mathrm{ is-operator-applicable-in I op
        using calculation
        unfolding are-all-operators-applicable-in-def
            SAS-Plus-Representation.is-operator-applicable-in-def list-all-iff
        by fastforce
    moreover have G\subseteqm
        using Cons.prems(2) calculation(2)
        unfolding is-operator-applicable-in-def
        by simp
    moreover have execute-parallel-plan-sas-plus I ([op] # embed \psi) =I
        using calculation(1)
        by fastforce
    ultimately show ?thesis
        by force
    qed
qed simp
```

```
lemma execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus-iii:
    assumes is-valid-problem-sas-plus \Psi
        and is-serial-solution-for-problem \Psi \psi
        and op\in set \psi
    shows are-operator-effects-consistent op op
proof -
    have op }\in\operatorname{set}((\Psi\mp@subsup{)}{O+}{*}
        using assms(2) assms(3)
        unfolding is-serial-solution-for-problem-def Let-def list-all-iff ListMem-iff
        by fastforce
    then have is-valid-operator-sas-plus \Psi op
        using is-valid-problem-sas-plus-then(2) assms(1, 3)
        by auto
    thus ?thesis
        unfolding are-operator-effects-consistent-def Let-def list-all-iff ListMem-iff
        using is-valid-operator-sas-plus-then(6)
        by fast
qed
lemma execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus-iv:
    fixes \Psi :: ('v,'d) sas-plus-problem
    assumes }\forallop\in\mathrm{ set }\psi.op\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O+}}{}
    shows }\forallops\in\operatorname{set}(embed \psi).\forallop\in set ops.op \in set ((\Psi\mp@subsup{)}{\mathcal{O+}}{}
proof -
    let ? }\mp@subsup{\psi}{}{\prime}=\mathrm{ embed }
    have nb: set? ' }\mp@subsup{}{\prime}{\prime}={[op]|op.op\in set \psi
        by (induction \psi; force)
    {
        fix ops
        assume ops \in set ? %'
        moreover obtain op where ops = [op] and op \in set ((\Psi) (\Psi)
            using assms(1) nb calculation
            by blast
        ultimately have }\forallop\in\mathrm{ set ops. op }\in\operatorname{set}((\Psi\mp@subsup{)}{\mp@subsup{\mathcal{O+}}{+}{\prime}}{}
            by fastforce
    }
    thus ?thesis..
qed
theorem execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus:
    assumes is-valid-problem-sas-plus \Psi
    and is-serial-solution-for-problem \Psi \psi
    shows is-parallel-solution-for-problem \Psi (embed \psi)
proof -
    let ?ops = sas-plus-problem.operators-of \Psi
    and ? }\mp@subsup{\psi}{}{\prime}=\mathrm{ embed }
    {
        thm execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus-ii[OF]
```

```
    have }(\Psi\mp@subsup{)}{G+}{}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-serial-plan-sas-plus (( }\Psi\mp@subsup{)}{I+}{})
        using assms(2)
        unfolding is-serial-solution-for-problem-def Let-def
        by simp
    moreover have }\forallop\in set \psi. are-operator-effects-consistent op op
    using execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus-iii[OF assms]..
    ultimately have }(\Psi\mp@subsup{)}{G+}{}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan-sas-plus (( }\Psi\mp@subsup{)}{I+}{})\mathrm{ ? ? }\mp@subsup{\psi}{}{\prime
        using execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus-ii
        by blast
    }
    moreover {
    have }\forallop\in set \psi.op set ((\Psi) () ()+ 
        using assms(2)
        unfolding is-serial-solution-for-problem-def Let-def list-all-iff ListMem-iff
        by fastforce
    hence }\forallops\in set ? \psi'. . \forallop \in set ops. op \in set ((\Psi) () ()+ 
        using execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus-iv
        by blast
}
ultimately show ?thesis
    unfolding is-parallel-solution-for-problem-def list-all-iff ListMem-iff Let-def
goal-of-def
    initial-of-def
    by fastforce
qed
lemma flattening-lemma-i:
    fixes \Psi :: ('v,'d) sas-plus-problem
    assumes }\forallops\in\mathrm{ set }\pi.\forallop\in\mathrm{ set ops.op }\in\mathrm{ set (( }\Psi\mp@subsup{)}{\mp@subsup{\mathcal{O+}}{+}{\prime}}{}
    shows }\forallop\in\operatorname{set}(\mathrm{ concat }\pi).op\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O+}}{}
proof -
    {
        fix op
        assume op \in set (concat \pi)
        moreover have op \in(\bigcupops \in set \pi. set ops)
            using calculation
            unfolding set-concat.
        then obtain ops where ops\in set \pi}\mathrm{ and op f set ops
            using UN-iff
            by blast
        ultimately have op set ((\Psi) (\mathcal{O+})
            using assms
            by blast
    }
    thus ?thesis..
qed
lemma flattening-lemma-ii:
    fixes I :: ('variable, 'domain) state
```

```
    assumes }\forallops\in\mathrm{ set }\psi.\existsop.ops=[op]^is-valid-operator-sas-plus \Psi o
    and G}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan-sas-plus I }
    shows }G\mp@subsup{\subseteq}{m}{}\mathrm{ execute-serial-plan-sas-plus I (concat }\psi\mathrm{ )
proof -
    show ?thesis
    using assms
    proof (induction \psi arbitrary:I)
        case (Cons ops \psi)
        obtain op where ops-is:ops=[op] and is-valid-op:is-valid-operator-sas-plus
\Psi op
            using Cons.prems(1)
            by auto
    then show ?case
            proof (cases are-all-operators-applicable-in I ops)
                case True
                let ?J = execute-parallel-operator-sas-plus I [op]
                and ?J' = execute-operator-sas-plus I op
            have n\mp@subsup{b}{1}{}}\mathrm{ : is-operator-applicable-in I op
                using True ops-is
                    unfolding are-all-operators-applicable-in-def is-operator-applicable-in-def
                list-all-iff
                by force
            have n\mp@subsup{b}{2}{}}\mathrm{ : are-operator-effects-consistent op op
                unfolding are-operator-effects-consistent-def list-all-iff Let-def
                using is-valid-operator-sas-plus-then(6)[OF is-valid-op]
                by blast
            have are-all-operator-effects-consistent ops
                using ops-is
                    unfolding are-all-operator-effects-consistent-def list-all-iff
                    using n\mp@subsup{b}{2}{}
                by force
            moreover have G\subseteq}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan-sas-plus ?J }
                    using Cons.prems(2) True calculation ops-is
                    by fastforce
            moreover have execute-serial-plan-sas-plus I (concat (ops # \psi))
                = execute-serial-plan-sas-plus ?J' (concat \psi)
                using ops-is nb i is-operator-applicable-in-def
                by simp
            moreover have ? J = ? J'
                using execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus-i[OF
nb 1 nb [
                    by simp
            ultimately show ?thesis
                    using Cons.IH[of ?J] Cons.prems(1)
                    by force
        next
            case False
            moreover have G\subseteqm}
```

```
            using Cons.prems(2) calculation
            by fastforce
            moreover {
                have \negis-operator-applicable-in I op
                    using False ops-is
                    unfolding are-all-operators-applicable-in-def
                    is-operator-applicable-in-def list-all-iff
                    by force
                    moreover have execute-serial-plan-sas-plus I (concat (ops # \psi))
                        = execute-serial-plan-sas-plus I (op # concat \psi)
                    using ops-is
                    by force
                        ultimately have execute-serial-plan-sas-plus I (concat (ops #\psi))=I
                    using False
                    unfolding is-operator-applicable-in-def
                    by fastforce
            }
            ultimately show ?thesis
                by argo
            qed
    qed force
qed
lemma flattening-lemma:
    assumes is-valid-problem-sas-plus \Psi
        and }\forallops\in set \psi.\existsop.ops=[op
        and is-parallel-solution-for-problem \Psi \psi
    shows is-serial-solution-for-problem \Psi (concat \psi)
proof -
    let ? }\mp@subsup{\psi}{}{\prime}=\mathrm{ concat }
    {
        have }\forallops\in set \psi. \forallop \in set ops.op \in set ((\Psi) () O+)
            using assms(3)
            unfolding is-parallel-solution-for-problem-def list-all-iff ListMem-iff
            by force
    hence }\forallop\in\mathrm{ set ? }\mp@subsup{\psi}{}{\prime}.op\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O+}}{}
            using flattening-lemma-i
            by blast
    }
    moreover {
    {
    fix ops
    assume ops \in set \psi
    moreover obtain op where ops=[op]
            using assms(2) calculation
            by blast
            moreover have op \in set ((\Psi)}\mp@subsup{)}{\mathcal{O+}}{}
                using assms(3) calculation
                    unfolding is-parallel-solution-for-problem-def list-all-iff ListMem-iff
```

```
            by force
    moreover have is-valid-operator-sas-plus \Psi op
        using assms(1) calculation(3)
        unfolding is-valid-problem-sas-plus-def Let-def list-all-iff
            ListMem-iff
        by simp
        ultimately have \existsop.ops=[op]^is-valid-operator-sas-plus \Psi op
        by blast
    }
    moreover have }(\Psi\mp@subsup{)}{G+}{}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan-sas-plus (( }\Psi\mp@subsup{)}{I+}{})
        using assms(3)
        unfolding is-parallel-solution-for-problem-def
        by fastforce
    ultimately have }(\Psi\mp@subsup{)}{G+}{}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-serial-plan-sas-plus (( }\Psi\mp@subsup{)}{I+}{})\mathrm{ ? ? ' '
    using flattening-lemma-ii
    by blast
}
ultimately show is-serial-solution-for-problem \Psi ? \psi '
    unfolding is-serial-solution-for-problem-def list-all-iff ListMem-iff
    by fastforce
qed
end
```


### 5.4 Auxiliary lemmata on SAS+

context
begin

- Relate the locale definition range-of with its corresponding implementation for valid operators and given an effect ( $v, a)$.
lemma is-valid-operator-sas-plus-then-range-of-sas-plus-op-is-set-range-of-op:
assumes is-valid-operator-sas-plus $\Psi$ op
and $(v, a) \in \operatorname{set}($ precondition-of op) $\vee(v, a) \in \operatorname{set}($ effect-of op)
shows $\left(\mathcal{R}_{+} \Psi v\right)=$ set (the (sas-plus-problem.range-of $\left.\Psi v\right)$ )
proof -
consider $(A)(v, a) \in \operatorname{set}($ precondition-of op)
| $(B) \quad(v, a) \in \operatorname{set}($ effect-of op)
using assms(2)..
thus ?thesis
proof (cases)
case $A$
then have $\left(\mathcal{R}_{+} \Psi v\right) \neq\{ \}$ and $a \in \mathcal{R}_{+} \Psi v$
using assms
unfolding range-of-def
using is-valid-operator-sas-plus-then(2)
by fast+
thus ?thesis
unfolding range-of'-def option.case-eq-if
by auto

```
    next
        case B
        then have (\mp@subsup{\mathcal{R}}{+}{}\Psiv)\not={} and a\in\mathcal{R}
        using assms
        unfolding range-of-def
        using is-valid-operator-sas-plus-then(4)
        by fast+
    thus ?thesis
        unfolding range-of'-def option.case-eq-if
        by auto
    qed
qed
lemma set-the-range-of-is-range-of-sas-plus-if:
    fixes \Psi :: ('v,'d) sas-plus-problem
    assumes is-valid-problem-sas-plus \Psi
        v\in set ((\Psi)}\mp@subsup{\mathcal{V}}{+}{\prime}
```



```
proof-
    have v\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{V}+}{+}
        using assms(2)
        unfolding variables-of-def.
    moreover have (\mathcal{R}+\Psiv)\not={}
        using assms(1) calculation is-valid-problem-sas-plus-then(1)
        by blast
    moreover have sas-plus-problem.range-of \Psi v = None
        and sas-plus-problem.range-of \Psiv}=\mathrm{ Some []
        using calculation(2) range-of-not-empty
        unfolding range-of-def
        by fast+
    ultimately show ?thesis
        unfolding option.case-eq-if range-of'-def
        by force
qed
lemma sublocale-sas-plus-finite-domain-representation-ii:
    fixes \Psi::('v,'d) sas-plus-problem
    assumes is-valid-problem-sas-plus \Psi
    shows }\forallv\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{V}+}{+})\cdot(\mp@subsup{\mathcal{R}}{+}{}\Psiv)\not={
        and }\forallop\in\mathrm{ set (( }\Psi\mp@subsup{)}{\mp@subsup{\mathcal{O}}{+}{+}}{)}). is-valid-operator-sas-plus \Psio
        and dom ((\Psi) I+ ) = set ((\Psi) (V+)
        and}\forallv\in\operatorname{dom}((\Psi\mp@subsup{)}{I+}{\prime})\mathrm{ . the }(((\Psi\mp@subsup{)}{I+}{+})v)\in\mp@subsup{\mathcal{R}}{+}{}\Psi
        and dom ((\Psi) (G+)\subseteq set ((\Psi)
        and}\forallv\in\operatorname{dom}((\Psi\mp@subsup{)}{G+}{*})\mathrm{ . the }(((\Psi\mp@subsup{)}{G+}{*})v)\in\mp@subsup{\mathcal{R}}{+}{}\Psi
    using is-valid-problem-sas-plus-then[OF assms]
    by auto
end
```


## 6 SAS+/STRIPS Equivalence

The following part is concerned with showing the equivalent expressiveness of SAS+ and STRIPS as discussed in ??.

### 6.1 Translation of SAS+ Problems to STRIPS Problems

definition possible-assignments-for
:: ('variable, 'domain) sas-plus-problem $\Rightarrow$ 'variable $\Rightarrow$ ('variable $\times$ 'domain) list
where possible-assignments-for $\Psi v \equiv[(v, a) . a \leftarrow$ the (range-of $\Psi v)]$
definition all-possible-assignments-for
:: ('variable, 'domain) sas-plus-problem $\Rightarrow$ ('variable $\times$ 'domain) list
where all-possible-assignments-for $\Psi$
$\equiv$ concat [possible-assignments-for $\Psi v . v \leftarrow$ variables-of $\Psi]$
definition state-to-strips-state
:: ('variable, 'domain) sas-plus-problem
$\Rightarrow$ ('variable, 'domain) state
$\Rightarrow$ ('variable, 'domain) assignment strips-state
( $\varphi_{S}-$ - 99 )
where state-to-strips-state $\Psi s$

$$
\begin{aligned}
& \equiv \text { let defined }=\text { filter }(\lambda v . \text { s } v \neq \text { None })(\text { variables-of } \Psi) \text { in } \\
& \text { map-of }(\text { map }(\lambda(v, a) .((v, a), \text { the }(s v)=a)) \\
& \quad(\text { concat }[\text { possible-assignments-for } \Psi v . v \leftarrow \text { defined }]))
\end{aligned}
$$

definition sasp-op-to-strips
:: ('variable, 'domain) sas-plus-problem
$\Rightarrow$ ('variable, 'domain) sas-plus-operator
$\Rightarrow$ ('variable, 'domain) assignment strips-operator
( $\varphi_{O}$ - - 99 )
where sasp-op-to-strips $\Psi$ op $\equiv$ let
pre $=$ precondition-of op
; add $=$ effect-of op
; delete $=\left[\left(v, a^{\prime}\right) .(v, a) \leftarrow\right.$ effect-of op, $a^{\prime} \leftarrow$ filter $((\neq) a)$ (the (range-of $\Psi$
$v)$ )]
in STRIPS-Representation.operator-for pre add delete
definition sas-plus-problem-to-strips-problem
:: ('variable, 'domain) sas-plus-problem $\Rightarrow$ ('variable, 'domain) assignment strips-problem
( $\varphi$ - 99 )
where sas-plus-problem-to-strips-problem $\Psi \equiv$ let
$v s=[$ as. $v \leftarrow$ variables-of $\Psi, a s \leftarrow($ possible-assignments-for $\Psi) v]$
; ops $=$ map $($ sasp-op-to-strips $\Psi)($ operators-of $\Psi)$
$; I=$ state-to-strips-state $\Psi$ (initial-of $\Psi)$
; $G=$ state-to-strips-state $\Psi$ (goal-of $\Psi)$
in STRIPS-Representation.problem-for vs ops IG
definition sas-plus-parallel-plan-to-strips-parallel-plan
:: ('variable, 'domain) sas-plus-problem
$\Rightarrow$ ('variable, 'domain) sas-plus-parallel-plan
$\Rightarrow$ ('variable $\times$ 'domain) strips-parallel-plan
( $\varphi_{P}-$ - 99 )
where sas-plus-parallel-plan-to-strips-parallel-plan $\Psi \psi$

$$
\equiv[[\text { sasp-op-to-strips } \Psi \text { op. op } \leftarrow o p s] . \text { ops } \leftarrow \psi]
$$

definition strips-state-to-state
:: ('variable, 'domain) sas-plus-problem
$\Rightarrow$ ('variable, 'domain) assignment strips-state
$\Rightarrow$ ('variable, 'domain) state
( $\varphi_{S}{ }^{-1}-$ - 99$)$
where strips-state-to-state $\Psi s$
$\equiv$ map-of $($ filter $(\lambda(v, a) . s(v, a)=$ Some True) (all-possible-assignments-for $\Psi)$ )
definition strips-op-to-sasp
:: ('variable, 'domain) sas-plus-problem
$\Rightarrow$ ('variable $\times$ 'domain) strips-operator
$\Rightarrow$ ('variable, 'domain) sas-plus-operator
( $\varphi_{O}{ }^{-1}-$ - 99 )
where strips-op-to-sasp $\Psi$ op
$\equiv$ let
precondition $=$ strips-operator.precondition-of op
; effect $=$ strips-operator.add-effects-of op
in \ precondition-of $=$ precondition, effect-of $=$ effect )
definition strips-parallel-plan-to-sas-plus-parallel-plan
:: ('variable, 'domain) sas-plus-problem
$\Rightarrow$ ('variable $\times$ 'domain) strips-parallel-plan
$\Rightarrow$ ('variable, 'domain) sas-plus-parallel-plan
$\left(\varphi_{P}{ }^{-1}-\right.$ - 99$)$
where strips-parallel-plan-to-sas-plus-parallel-plan $\Pi \pi$

$$
\equiv[[\text { strips-op-to-sasp } \Pi \text { op. op } \leftarrow \text { ops }] . \text { ops } \leftarrow \pi]
$$

To set up the equivalence proof context, we declare a common locale for
both the STRIPS and SAS+ formalisms and make it a sublocale of both locale as well as . The declaration itself is omitted for brevity since it basically just joins locales and while renaming the locale parameter to avoid name clashes. The sublocale proofs are shown below. ${ }^{5}$

```
definition range-of-strips \(\Pi x \equiv\{\) True, False \(\}\)
context
begin
- Set-up simp rules.
lemma \([\) simp \(]\) :
    \((\varphi \Psi)=(\) let
        \(v s=[\) as. \(v \leftarrow\) variables-of \(\Psi, a s \leftarrow(\) possible-assignments-for \(\Psi) v]\)
        ; ops \(=\) map (sasp-op-to-strips \(\Psi)(\) operators-of \(\Psi)\)
        \(; I=\) state-to-strips-state \(\Psi\) (initial-of \(\Psi)\)
        ; \(G=\) state-to-strips-state \(\Psi\) (goal-of \(\Psi\) )
    in STRIPS-Representation.problem-for vs ops \(I G\) )
    and \(\left(\varphi_{S} \Psi s\right)\)
    \(=(\) let defined \(=\) filter \((\lambda v\). s \(v \neq\) None \()(\) variables-of \(\Psi)\) in
        map-of \((\operatorname{map}(\lambda(v, a) .((v, a)\), the \((s v)=a))\)
            (concat [possible-assignments-for \(\Psi\) v. \(v \leftarrow\) defined \(]))\) )
    and \(\left(\varphi_{O} \Psi o p\right)\)
    \(=\) (let
            pre \(=\) precondition-of op
            ; add \(=\) effect-of op
            ; delete \(=\left[\left(v, a^{\prime}\right) .(v, a) \leftarrow\right.\) effect-of op, \(a^{\prime} \leftarrow\) filter \(((\neq) a)\) (the (range-of \(\Psi\)
v) )]
    in STRIPS-Representation.operator-for pre add delete)
and \(\left(\varphi_{P} \Psi \psi\right)=\left[\left[\varphi_{O} \Psi\right.\right.\) op. op \(\left.\leftarrow o p s\right]\). ops \(\left.\leftarrow \psi\right]\)
and \(\left(\varphi_{S}{ }^{-1} \Psi s^{\prime}\right)=\) map-of (filter \(\left(\lambda(v, a) . s^{\prime}(v, a)=\right.\) Some True)
    (all-possible-assignments-for \(\Psi)\) )
and \(\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)=(\) let
                precondition \(=\) strips-operator.precondition-of op \({ }^{\prime}\)
                ; effect \(=\) strips-operator.add-effects-of op \({ }^{\prime}\)
        in ( precondition-of \(=\) precondition, effect-of \(=\) effect \(D)\)
    and \(\left(\varphi_{P}{ }^{-1} \Psi \pi\right)=\left[\left[\varphi_{O}^{-1} \Psi\right.\right.\) op. op \(\left.\leftarrow o p s\right]\). ops \(\left.\leftarrow \pi\right]\)
    unfolding
    SAS-Plus-STRIPS.sas-plus-problem-to-strips-problem-def
    sas-plus-problem-to-strips-problem-def
    SAS-Plus-STRIPS.state-to-strips-state-def
    state-to-strips-state-def
```

[^5]```
    SAS-Plus-STRIPS.sasp-op-to-strips-def
    sasp-op-to-strips-def
    SAS-Plus-STRIPS.sas-plus-parallel-plan-to-strips-parallel-plan-def
    sas-plus-parallel-plan-to-strips-parallel-plan-def
    SAS-Plus-STRIPS.strips-state-to-state-def
    strips-state-to-state-def
    SAS-Plus-STRIPS.strips-op-to-sasp-def
    strips-op-to-sasp-def
    SAS-Plus-STRIPS.strips-parallel-plan-to-sas-plus-parallel-plan-def
    strips-parallel-plan-to-sas-plus-parallel-plan-def
    by blast+
lemmas [simp] = range-of'-def
lemma is-valid-problem-sas-plus-dom-sas-plus-problem-range-of:
    assumes is-valid-problem-sas-plus \Psi
    shows }\forallv\in\operatorname{set}((\Psi)\mp@subsup{\mathcal{V}}{+}{}).v\indom (sas-plus-problem.range-of \Psi
    using assms(1) is-valid-problem-sas-plus-then(1)
    unfolding is-valid-problem-sas-plus-def
    by (meson domIff list.pred-set)
lemma possible-assignments-for-set-is:
    assumes v\indom (sas-plus-problem.range-of \Psi)
    shows set (possible-assignments-for \Psi v)
    ={(v,a)|a.a\in\mathcal{R}+\Psiv}
proof -
    have sas-plus-problem.range-of \Psi v}=\mathrm{ None
        using assms(1)
        by auto
    thus ?thesis
        unfolding possible-assignments-for-def
        by fastforce
qed
lemma all-possible-assignments-for-set-is:
    assumes }\forallv\in\operatorname{set}((\Psi)\mp@subsup{\mathcal{V}}{+}{})\mathrm{ . range-of }\Psiv\not=Non
    shows set (all-possible-assignments-for \Psi)
    =(\bigcupv\in\operatorname{set}((\Psi)\mp@subsup{\mathcal{V}}{+}{}).{(v,a)|a.a\in\mathcal{R}+\Psiv})
proof -
    let ?vs= variables-of }
    have set (all-possible-assignments-for }\Psi\mathrm{ ) =
        (U(set'(\lambdav.map (\lambda(v,a). (v,a))(possible-assignments-for \Psi v))' set ?vs))
        unfolding all-possible-assignments-for-def set-concat
        using set-map
        by auto
    also have ... = (\bigcup((\lambdav. set (possible-assignments-for \Psi v))'set ?vs))
        using image-comp set-map
        by simp
```

```
    also have \(\ldots=\left(\bigcup\left(\left(\lambda v .\left\{(v, a) \mid a . a \in \mathcal{R}_{+} \Psi v\right\}\right)\right.\right.\) ' set ? \(\left.\left.v s\right)\right)\)
        using possible-assignments-for-set-is assms
    by fastforce
    finally show ?thesis
    by force
qed
lemma state-to-strips-state-dom-is-i[simp]:
    assumes \(\forall v \in \operatorname{set}\left((\Psi) \mathcal{V}_{+}\right) . v \in\) dom (sas-plus-problem.range-of \(\left.\Psi\right)\)
    shows set (concat
        [possible-assignments-for \(\Psi v . v \leftarrow\) filter \((\lambda v . s v \neq\) None) (variables-of \(\Psi)]\) )
    \(=\left(\bigcup v \in\left\{v \mid v . v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right) \wedge s v \neq\right.\right.\) None \(\}\).
        \(\left.\left\{(v, a) \mid a . a \in \mathcal{R}_{+} \Psi v\right\}\right)\)
proof -
    let \(? v s=\) variables-of \(\Psi\)
    let ?defined \(=\) filter \((\lambda v . s v \neq\) None \()\) ?vs
    let ?l = concat [possible-assignments-for \(\Psi v . v \leftarrow\) ?defined]
    have nb: set ?defined \(=\left\{v \mid v . v \in \operatorname{set}\left((\Psi)_{\mathcal{V}+}\right) \wedge s v \neq\right.\) None \(\}\)
        unfolding set-filter
        by force
    have set ?l \(=\bigcup(\) set'set (map (possible-assignments-for \(\Psi)\) ?defined \())\)
        unfolding set-concat image-Union
        by blast
    also have \(\ldots=\bigcup\) (set ' (possible-assignments-for \(\Psi\) )' set ?defined)
        unfolding set-map
        by blast
    also have \(\ldots=(\bigcup v \in\) set ?defined. set (possible-assignments-for \(\Psi v))\)
        by blast
    also have \(\ldots=\left(\bigcup v \in\left\{v \mid v . v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right) \wedge s v \neq\right.\right.\) None \(\}\).
        set (possible-assignments-for \(\Psi v)\) )
        using \(n b\)
        by argo
    finally show ?thesis
        using possible-assignments-for-set-is
        is-valid-problem-sas-plus-dom-sas-plus-problem-range-of assms(1)
    by fastforce
qed
lemma state-to-strips-state-dom-is:
- NOTE A transformed state is defined on all possible assignments for all variables defined in the original state.
assumes is-valid-problem-sas-plus \(\Psi\)
shows \(\operatorname{dom}\left(\varphi_{S} \Psi s\right)\)
```

```
    \(=\left(\bigcup v \in\left\{v \mid v . v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right) \wedge s v \neq\right.\right.\) None \(\}\).
```

    \(=\left(\bigcup v \in\left\{v \mid v . v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right) \wedge s v \neq\right.\right.\) None \(\}\).
    \(\left.\left\{(v, a) \mid a . a \in \mathcal{R}_{+} \Psi v\right\}\right)\)
    proof -
let $? v s=$ variables - of $\Psi$
let ?l = concat [possible-assignments-for $\Psi v . v \leftarrow$ filter ( $\lambda v$. s $v \neq$ None) ?vs]
have $n b: \forall v \in \operatorname{set}\left((\Psi)_{\mathcal{V}+}\right) . v \in \operatorname{dom}$ (sas-plus-problem.range-of $\Psi$ )

```
using is-valid-problem-sas-plus-dom-sas-plus-problem-range-of assms(1) by fastforce
have \(\operatorname{dom}\left(\varphi_{S} \Psi s\right)=f s t ' \operatorname{set}(\operatorname{map}(\lambda(v, a) .((v, a)\), the \((s v)=a))\) ?l) unfolding state-to-strips-state-def
SAS-Plus-STRIPS.state-to-strips-state-def
using dom-map-of-conv-image-fst[of map \((\lambda(v, a) .((v, a)\), the \((s v)=a))\) ?l]
by presburger
also have \(\ldots=f_{s t} \cdot(\lambda(v, a) .((v, a)\), the \((s v)=a))\) 'set ?l
unfolding set-map
by blast
also have \(\ldots=(\lambda(v, a)\). fst \(((v, a)\), the \((s v)=a))\) ' set ?l
unfolding image-comp[of fst \(\lambda(v, a)\). \(((v, a)\), the \((s v)=a)]\) comp-apply \([o f\) \(f s t \lambda(v, a) .((v, a)\), the \((s v)=a)]\) prod.case-distrib
by blast
finally show ?thesis
unfolding state-to-strips-state-dom-is-i[OF nb]
by force
qed
corollary state-to-strips-state-dom-element-iff:
assumes is-valid-problem-sas-plus \(\Psi\)
shows \((v, a) \in \operatorname{dom}\left(\varphi_{S} \Psi s\right) \longleftrightarrow v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
\(\wedge s v \neq\) None
\(\wedge a \in \mathcal{R}_{+} \Psi v\)
proof -
let \(? v s=\) variables-of \(\Psi\)
and ? \(s^{\prime}=\varphi_{S} \Psi s\)
show ?thesis
proof (rule iffI)
assume \((v, a) \in \operatorname{dom}\left(\varphi_{S} \Psi s\right)\)
then have \(v \in\left\{v \mid v . v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right) \wedge s v \neq\right.\) None \(\}\)
and \(a \in \mathcal{R}_{+} \Psi v\)
unfolding state-to-strips-state-dom-is[OF assms(1)]
by force+
moreover have \(v \in\) set ?vs and \(s v \neq\) None using calculation(1)
by fastforce+
ultimately show
\(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right) \wedge s v \neq\) None \(\wedge a \in \mathcal{R}_{+} \Psi v\)
by force
next
assume \(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right) \wedge s v \neq\) None \(\wedge a \in \mathcal{R}_{+} \Psi v\)
then have \(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
and \(s v \neq\) None
and \(a\)-in-range-of-v: \(a \in \mathcal{R}_{+} \Psi v\)
by \(\operatorname{simp}+\)
then have \(v \in\left\{v \mid v . v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right) \wedge s v \neq\right.\) None \(\}\)
by force
thus \((v, a) \in \operatorname{dom}\left(\varphi_{S} \Psi s\right)\)
```

            unfolding state-to-strips-state-dom-is[OF assms(1)]
            using \(a\)-in-range-of-v
            by blast
    qed
    qed
lemma state-to-strips-state-range-is:
assumes is-valid-problem-sas-plus $\Psi$
and $(v, a) \in \operatorname{dom}\left(\varphi_{S} \Psi s\right)$
shows $\left(\varphi_{S} \Psi s\right)(v, a)=$ Some (the $\left.(s v)=a\right)$
proof -
let ?vs $=$ variables-of $\Psi$
let $? s^{\prime}=\varphi_{S} \Psi s$
and ?defined $=$ filter $(\lambda v$. s $v \neq$ None $)$ ?vs
let ?l = concat [possible-assignments-for $\Psi v . v \leftarrow$ ?defined]
have $v$-in-set-vs: $v \in$ set ?vs
and $s$-of-v-is-not-None: s $v \neq$ None
and $a$-in-range-of-v: $a \in \mathcal{R}_{+} \Psi v$
using assms(2)
unfolding state-to-strips-state-dom-is[OF assms(1)]
by fastforce+
moreover \{
have $\forall v \in \operatorname{set}\left((\Psi)_{\mathcal{V}+}\right) . v \in \operatorname{dom}$ (sas-plus-problem.range-of $\left.\Psi\right)$
using assms(1) is-valid-problem-sas-plus-then(1)
unfolding is-valid-problem-sas-plus-def
by fastforce
moreover have $(v, a) \in$ set ?l
unfolding state-to-strips-state-dom-is-i[OF calculation(1)]
using $s$-of-v-is-not-None $a$-in-range-of-v $v$-in-set-vs
by fastforce
moreover have set $? l \neq\{ \}$
using calculation
by fastforce
- TODO slow.
ultimately have $\left(\varphi_{S} \Psi s\right)(v, a)=$ Some (the $\left.(s v)=a\right)$
using map-of-from-function-graph-is-some-if [of
?l $(v, a) \lambda(v, a)$. the $(s v)=a]$
unfolding SAS-Plus-STRIPS.state-to-strips-state-def
state-to-strips-state-def Let-def case-prod-beta'
by fastforce
\}
thus ?thesis.
qed

```
- Show that a STRIPS state corresponding to a SAS+ state via transformation is consistent w.r.t. to the variable subset with same left component (i.e. the original SAS+ variable). This is the consistency notion corresponding to SAS+ consistency: i.e. if no two assignments with different values for the same variable
exist in the SAS+ state, then assigning the corresponding assignment both to True is impossible. Vice versa, if both are assigned to True then the assignment variables must be the same SAS+ variable/SAS+ value pair.
lemma state-to-strips-state-effect-consistent:
assumes is-valid-problem-sas-plus \(\Psi\)
and \((v, a) \in \operatorname{dom}\left(\varphi_{S} \Psi s\right)\)
and \(\left(v, a^{\prime}\right) \in \operatorname{dom}\left(\varphi_{S} \Psi s\right)\)
and \(\left(\varphi_{S} \Psi s\right)(v, a)=\) Some True
and \(\left(\varphi_{S} \Psi s\right)\left(v, a^{\prime}\right)=\) Some True
shows \((v, a)=\left(v, a^{\prime}\right)\)
proof -
have the \((s v)=a\) and the \((s v)=a^{\prime}\)
using state-to-strips-state-range-is \([\) OF \(\operatorname{assms}(1)] \operatorname{assms}(2,3,4,5)\)
by fastforce+
thus ?thesis
by argo
qed
lemma sasp-op-to-strips-set-delete-effects-is:
assumes is-valid-operator-sas-plus \(\Psi\) op
shows set (strips-operator.delete-effects-of \(\left.\left(\varphi_{O} \Psi o p\right)\right)\)
\(=\left(\bigcup(v, a) \in \operatorname{set}(\right.\) effect-of op \(\left.) .\left\{\left(v, a^{\prime}\right) \mid a^{\prime} . a^{\prime} \in\left(\mathcal{R}_{+} \Psi v\right) \wedge a^{\prime} \neq a\right\}\right)\)
proof -
let \(? D=\) range-of \(\Psi\)
and ?effect \(=\) effect-of op
let ?delete \(=\left[\left(v, a^{\prime}\right) .(v, a) \leftarrow\right.\) ?effect, \(a^{\prime} \leftarrow\) filter \(((\neq) a)(\) the \(\left.(? D v))\right]\)
\{
fix \(v a\)
assume \((v, a) \in\) set ?effect
then have \(\left(\mathcal{R}_{+} \Psi v\right)=\operatorname{set}(\) the \((? D v))\)
using assms
using is-valid-operator-sas-plus-then-range-of-sas-plus-op-is-set-range-of-op
by fastforce
hence set \((\) filter \(((\neq) a)(\) the \((? D v)))=\left\{a^{\prime} \in \mathcal{R}_{+} \Psi v \cdot a^{\prime} \neq a\right\}\)
unfolding set-filter
by blast
\} note \(n b=t h i s\)
\{
- TODO slow.
have set ? delete \(=\bigcup(\) set ' \((\lambda(v, a)\). map \((\) Pair \(v)(\) filter \(((\neq) a)(\) the \((? D v))))\)
' (set ?effect))
using set-concat
by \(\operatorname{simp}\)
also have \(\ldots=\bigcup((\lambda(v, a)\). Pair \(v ' \operatorname{set}(\) filter \(((\neq) a)(\) the \((? D v))))\)
'(set ?effect))
unfolding image-comp[of set] set-map
by auto
- TODO slow.
also have \(\ldots=\left(\bigcup(v, a) \in\right.\) set ?effect. Pair \(\left.v^{\prime}\left\{a^{\prime} \in \mathcal{R}_{+} \Psi v \cdot a^{\prime} \neq a\right\}\right)\) using \(n b\) by fast
finally have set ?delete \(=(\bigcup(v, a) \in\) set ?effect.
\(\left.\left\{\left(v, a^{\prime}\right) \mid a^{\prime} . a^{\prime} \in\left(\mathcal{R}_{+} \Psi v\right) \wedge a^{\prime} \neq a\right\}\right)\)
by blast
\}
thus ?thesis
unfolding SAS-Plus-STRIPS.sasp-op-to-strips-def sasp-op-to-strips-def Let-def
by force
qed
lemma sas-plus-problem-to-strips-problem-variable-set-is:
- The variable set of \(\Pi\) is the set of all possible assignments that are possible using the variables of \(\mathcal{V}\) and the corresponding domains.
assumes is-valid-problem-sas-plus \(\Psi\)
shows \(\operatorname{set}\left((\varphi \Psi)_{\mathcal{V}}\right)=\left(\bigcup v \in \operatorname{set}\left((\Psi)_{\mathcal{V}}\right) .\left\{(v, a) \mid a . a \in \mathcal{R}_{+} \Psi v\right\}\right)\)
proof -
let \(? \Pi=\varphi \Psi\)
and ?vs \(=\) variables-of \(\Psi\)
\{
have set (strips-problem.variables-of ? \(\Pi\) )
\(=\) set \([\) as. \(v \leftarrow\) ? vs, as \(\leftarrow\) possible-assignments-for \(\Psi v]\)
unfolding sas-plus-problem-to-strips-problem-def
SAS-Plus-STRIPS.sas-plus-problem-to-strips-problem-def
by force
also have \(\ldots=(\bigcup(\) set ' \((\lambda v\). possible-assignments-for \(\Psi v)\) 'set ?vs \())\)
using set-concat
by auto
also have \(\ldots=(\bigcup((\) set \(\circ\) possible-assignments-for \(\Psi)\) ' set ?vs \())\)
using image-comp[of set \(\lambda v\). possible-assignments-for \(\Psi v\) set ?vs]
by argo
finally have set (strips-problem.variables-of ? \(\Pi\) )
\(=(\bigcup v \in\) set ? \(v s\). set (possible-assignments-for \(\Psi v))\)
unfolding o-apply
by blast
\}
moreover have \(\forall v \in\) set ?vs. \(v \in\) dom (sas-plus-problem.range-of \(\Psi\) )
using is-valid-problem-sas-plus-dom-sas-plus-problem-range-of assms by force
ultimately show ?thesis
using possible-assignments-for-set-is
by force
qed
corollary sas-plus-problem-to-strips-problem-variable-set-element-iff:
assumes is-valid-problem-sas-plus \(\Psi\)
```

    shows }(v,a)\in\operatorname{set}((\varphi\Psi\mp@subsup{)}{\mathcal{V}}{})\longleftrightarrowv\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{V}+}{*})\wedgea\in\mp@subsup{\mathcal{R}}{+}{}\Psi
    unfolding sas-plus-problem-to-strips-problem-variable-set-is[OF assms]
    by fast
    lemma sasp-op-to-strips-effect-consistent:
assumes op = \mp@subsup{\varphi}{O}{}\Psio\mp@subsup{p}{}{\prime}
and o\mp@subsup{p}{}{\prime}\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O+}}{})
and is-valid-operator-sas-plus \Psi op'
shows (v,a)\in set (add-effects-of op) \longrightarrow(v,a)\not\in set (delete-effects-of op)
and (v,a)\in set (delete-effects-of op) \longrightarrow(v,a) \& set (add-effects-of op)
proof -
have nb: (\forall (v,a)\in set (effect-of o\mp@subsup{p}{}{\prime}).}\forall(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\operatorname{set}(\mathrm{ effect-of op').v}v\not=\mp@subsup{v}{}{\prime}\vee
= a}\mathrm{ )
using assms(3)
unfolding is-valid-operator-sas-plus-def
SAS-Plus-Representation.is-valid-operator-sas-plus-def list-all-iff ListMem-iff
Let-def
by argo
{
fix va
assume v-a-in-add-effects-of-op: (v,a) \in set (add-effects-of op)
have (v,a) \& set (delete-effects-of op)
proof (rule ccontr)
assume }\neg(v,a)\not\in\mathrm{ set (delete-effects-of op)
moreover have (v,a)\in
(U(v,\mp@subsup{a}{}{\prime})\in\operatorname{set}(\mathrm{ effect-of op'). { (v, a'')}
| a'. a'首 ( (\mathcal{R}}+\Psiv)^\mp@subsup{a}{}{\prime\prime}\not=\mp@subsup{a}{}{\prime}}
using calculation sasp-op-to-strips-set-delete-effects-is
assms
by blast
moreover obtain }\mp@subsup{a}{}{\prime}\mathrm{ where (v, a') f set (effect-of op') and a}\not=\mp@subsup{a}{}{\prime
using calculation
by blast
moreover have (v, a})\in\operatorname{set}(add-effects-of op
using assms(1) calculation(3)
unfolding sasp-op-to-strips-def
SAS-Plus-STRIPS.sasp-op-to-strips-def
Let-def
by fastforce
moreover have (v,a)\inset (effect-of op') and (v, a') \in set (effect-of op')
using assms(1) v-a-in-add-effects-of-op calculation(5)
unfolding sasp-op-to-strips-def
SAS-Plus-STRIPS.sasp-op-to-strips-def
Let-def
by force+
ultimately show False
using nb
by fast
qed

```
```

}
moreover {
fix }v
assume v-a-in-delete-effects-of-op: (v,a) \& set (delete-effects-of op)
have (v,a) \& set (add-effects-of op)
proof (rule ccontr)
assume}\neg(v,a)\not\in\mathrm{ set (add-effects-of op)
moreover have (v,a) \in set (add-effects-of op)
using calculation
by blast
moreover have (v,a)\in
(\bigcup(v,\mp@subsup{a}{}{\prime})\in\operatorname{set}(\mathrm{ effect-of op'). { (v, a'')}
| a'. a'音 \in(\mathcal{R}
using sasp-op-to-strips-set-delete-effects-is
nb assms(1, 3) v-a-in-delete-effects-of-op
by force
moreover obtain }\mp@subsup{a}{}{\prime}\mathrm{ where (v, a') { set (effect-of op') and a}=\mp@subsup{a}{}{\prime
using calculation
by blast
moreover have (v, a')\in set (add-effects-of op)
using assms(1) calculation(4)
unfolding sasp-op-to-strips-def
SAS-Plus-STRIPS.sasp-op-to-strips-def
Let-def
by fastforce
moreover have (v,a)\in set (effect-of o\mp@subsup{p}{}{\prime}) and (v, a')\in set (effect-of op')
using assms(1) calculation(2, 6)
unfolding sasp-op-to-strips-def
SAS-Plus-STRIPS.sasp-op-to-strips-def Let-def
by force+
ultimately show False
using nb
by fast
qed
}
ultimately show (v,a)\in set (add-effects-of op)
\longrightarrow ( v , a ) \notin set (delete-effects-of op)
and (v,a)\in set (delete-effects-of op)
\longrightarrow ( v , a ) \notin s e t ~ ( a d d - e f f e c t s - o f ~ o p )
by blast+
qed
lemma is-valid-problem-sas-plus-then-strips-transformation-too-iii:
assumes is-valid-problem-sas-plus \Psi
shows list-all (is-valid-operator-strips ( }\varphi\Psi)\mathrm{ )
(strips-problem.operators-of (\varphi\Psi))
proof -
let ?\Pi=\varphi\Psi
let ?vs = strips-problem.variables-of ?\Pi

```

\section*{\{}
fix op
assume \(o p \in\) set (strips-problem.operators-of ? \(\Pi\) )
- TODO slow.
then obtain \(o p^{\prime}\)
where \(o p-i s\) : \(o p=\varphi_{O} \Psi o p^{\prime}\)
and \(o p^{\prime}\)-in-operators: \(o p^{\prime} \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)
unfolding SAS-Plus-STRIPS.sas-plus-problem-to-strips-problem-def sas-plus-problem-to-strips-problem-def sasp-op-to-strips-def
by auto
then have is-valid-op': is-valid-operator-sas-plus \(\Psi o p^{\prime}\)
using sublocale-sas-plus-finite-domain-representation-ii(2)[OF assms]
by blast
moreover \{
fix \(v a\)
assume \((v, a) \in\) set (strips-operator.precondition-of op)
- TODO slow.
then have \((v, a) \in \operatorname{set}\) (sas-plus-operator.precondition-of op \({ }^{\prime}\) )
using op-is
unfolding SAS-Plus-STRIPS.sasp-op-to-strips-def
sasp-op-to-strips-def
by force
moreover have \(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
using is-valid-op' calculation
using is-valid-operator-sas-plus-then(1)
by fastforce
moreover have \(a \in \mathcal{R}_{+} \Psi v\)
using is-valid-op \({ }^{\prime}\) calculation(1)
using is-valid-operator-sas-plus-then(2)
by fast
ultimately have \((v, a) \in\) set ?vs
using sas-plus-problem-to-strips-problem-variable-set-element-iff[OF assms(1)] by force
\}
moreover \{
fix \(v a\)
assume \((v, a) \in \operatorname{set}\) (strips-operator.add-effects-of op)
then have \((v, a) \in\) set (effect-of op \({ }^{\prime}\) )
using op-is
unfolding SAS-Plus-STRIPS.sasp-op-to-strips-def
sasp-op-to-strips-def
by force
then have \(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)and \(a \in \mathcal{R}_{+} \Psi v\)
using is-valid-operator-sas-plus-then is-valid-op \({ }^{\prime}\)
by fastforce+
hence \((v, a) \in\) set ? vs
using sas-plus-problem-to-strips-problem-variable-set-element-iff[OF assms(1)]
by force
```

}
moreover {
fix }v\mp@subsup{a}{}{\prime
assume v-a'-in-delete-effects:(v, a')\in set (strips-operator.delete-effects-of
op)
moreover have set (strips-operator.delete-effects-of op)
= (U(v,a) \in set (effect-of op').
{(v,\mp@subsup{a}{}{\prime})|\mp@subsup{a}{}{\prime}.\mp@subsup{a}{}{\prime}\in(\mp@subsup{\mathcal{R}}{+}{}\Psiv)\wedge\mp@subsup{a}{}{\prime}\not=a})
using sasp-op-to-strips-set-delete-effects-is[OF is-valid-op ]
op-is
by simp
- TODO slow.
ultimately obtain a
where (v,a) \in set (effect-of op')
and }\mp@subsup{a}{}{\prime}-in:\mp@subsup{a}{}{\prime}\in{\mp@subsup{a}{}{\prime}\in\mp@subsup{\mathcal{R}}{+}{}\Psiv. v. a'\not=a
by blast
moreover have is-valid-operator-sas-plus \Psiop'
using op'-in-operators assms(1)
is-valid-problem-sas-plus-then(2)
by blast
moreover have v\in set ((\Psi)\mp@subsup{\mathcal{V}}{+}{})
using is-valid-operator-sas-plus-then calculation(1, 3)
by fast
moreover have a}\mp@subsup{a}{}{\prime}\in\mp@subsup{\mathcal{R}}{+}{\Psi}
using a'-in
by blast
ultimately have ( }v,\mp@subsup{a}{}{\prime})\in\mathrm{ set ?vs
using sas-plus-problem-to-strips-problem-variable-set-element-iff[OF assms(1)]
by force
}
ultimately have set (strips-operator.precondition-of op) \subseteqset ?vs
^set (strips-operator.add-effects-of op) \subseteq set ?vs
set (strips-operator.delete-effects-of op) \subseteq set ?vs
\wedge(\forallv\inset (add-effects-of op).v\not\in set (delete-effects-of op))
\wedge(\forallv\inset (delete-effects-of op).v}\not\in\mathrm{ set (add-effects-of op))
using sasp-op-to-strips-effect-consistent[OF
op-is op'-in-operators is-valid-op]
by fast+
}
thus ?thesis
unfolding is-valid-operator-strips-def STRIPS-Representation.is-valid-operator-strips-def
list-all-iff ListMem-iff Let-def
by blast
qed
lemma is-valid-problem-sas-plus-then-strips-transformation-too-iv:
assumes is-valid-problem-sas-plus \Psi
shows }\forallx.((\varphi\Psi\mp@subsup{)}{I}{})x\not=Non

```
```

    \(\longleftrightarrow\) ListMem \(x\) (strips-problem.variables-of \((\varphi \Psi))\)
    proof -
let ?vs $=$ variables-of $\Psi$
and ? $I=$ initial-of $\Psi$
and $? \Pi=\varphi \Psi$
let ?vs ${ }^{\prime}=$ strips-problem.variables-of ? $\Pi$
and $? I^{\prime}=$ strips-problem.initial-of ? $\Pi$
\{
fix $x$
have ? $I^{\prime} x \neq$ None $\longleftrightarrow$ ListMem $x$ ?vs'
proof (rule iffI)
assume $I^{\prime}$-of- $x$-is-not-None: ? $I^{\prime} x \neq$ None
then have $x \in \operatorname{dom} ? I^{\prime}$
by blast
moreover obtain $v a$ where $x$-is: $x=(v, a)$
by fastforce
ultimately have $(v, a) \in \operatorname{dom} ? I^{\prime}$
by blast
then have $v \in$ set ? vs
and ?I $v \neq$ None
and $a \in \mathcal{R}_{+} \Psi v$
using state-to-strips-state-dom-element-iff[OF assms(1), of va ?I]
unfolding sas-plus-problem-to-strips-problem-def
SAS-Plus-STRIPS.sas-plus-problem-to-strips-problem-def
state-to-strips-state-def
SAS-Plus-STRIPS.state-to-strips-state-def
by $\operatorname{simp}+$
thus ListMem $x$ ? vs ${ }^{\prime}$
unfolding ListMem-iff
using sas-plus-problem-to-strips-problem-variable-set-element-iff[OF assms(1)]
$x$-is
by auto
next
assume list-mem-x-vs': ListMem $x$ ? $v s^{\prime}$
then obtain $v a$ where $x$-is: $x=(v, a)$
by fastforce
then have $(v, a) \in$ set ? $v s^{\prime}$
using list-mem-x-vs'
unfolding ListMem-iff
by blast
then have $v \in$ set ? $v s$ and $a \in \mathcal{R}_{+} \Psi v$
using sas-plus-problem-to-strips-problem-variable-set-element-iff[OF assms(1)]
by force+
moreover have ?I $v \neq$ None
using is-valid-problem-sas-plus-then(3) assms(1) calculation(1)
by auto
ultimately have $(v, a) \in \operatorname{dom} ? I^{\prime}$
using state-to-strips-state-dom-element-iff $[O F \operatorname{assms}(1)$, of $v$ a ? I]

```
```

            unfolding SAS-Plus-STRIPS.sas-plus-problem-to-strips-problem-def
                    sas-plus-problem-to-strips-problem-def
                SAS-Plus-STRIPS.state-to-strips-state-def
                state-to-strips-state-def
            by force
            thus ?I' }x\not=\mathrm{ None
            using x-is
            by fastforce
        qed
    }
    thus ?thesis
        by simp
    qed
private lemma is-valid-problem-sas-plus-then-strips-transformation-too-v:
assumes is-valid-problem-sas-plus \Psi
shows }\forallx.((\varphi\Psi\mp@subsup{)}{G}{})x\not=N\mathrm{ None
\longrightarrow ListMem x (strips-problem.variables-of ( \varphi \Psi ) )
proof -
let ?vs = variables-of }
and ?D = range-of }
and ?G= goal-of \Psi
let ?\Pi=\varphi\Psi
let ?vs' = strips-problem.variables-of ?\Pi
and ? G' = strips-problem.goal-of ?\Pi
have nb:?G' = \varphi S \Psi ?G
by simp
{
fix }
assume ?G' }x\not=Non
moreover obtain va}\mathrm{ where }x=(v,a
by fastforce
moreover have (v,a)\indom ?G'
using domIff calculation(1, 2)
by blast
moreover have v\in set ?vs and }a\in\mp@subsup{\mathcal{R}}{+}{}\Psi
using state-to-strips-state-dom-is[OF assms(1), of ?G] nb calculation(3)
by auto+
ultimately have }x\in\mathrm{ set ?vs'
using sas-plus-problem-to-strips-problem-variable-set-element-iff[OF assms(1)]
by auto
}
thus ?thesis
unfolding ListMem-iff
by simp
qed

```

We now show that given \(\Psi\) is a valid SASPlus problem, then \(\Pi \equiv \varphi \Psi\) is a valid STRIPS problem as well. The proof unfolds the definition of
is-valid-problem-strips and then shows each of the conjuncts for \(\Pi\). These are:
- \(\Pi\) has at least one variable;
- \(\Pi\) has at least one operator;
- all operators are valid STRIPS operators;
- \(\Pi_{I}\) is defined for all variables in \(\Pi_{\mathcal{V}}\); and finally,
- if \(\left(\Pi_{G}\right) x\) is defined, then \(x\) is in \(\Pi_{\mathcal{V}}\).

\section*{theorem}
is-valid-problem-sas-plus-then-strips-transformation-too:
assumes is-valid-problem-sas-plus \(\Psi\)
shows is-valid-problem-strips \((\varphi \Psi)\)
proof -
let ? \(\Pi=\varphi \Psi\)
have list-all (is-valid-operator-strips \((\varphi \Psi)\) )
(strips-problem.operators-of ( \(\varphi \Psi\) ))
using is-valid-problem-sas-plus-then-strips-transformation-too-iii[OF assms].
moreover have \(\forall x\). \(\left(\left((\varphi \Psi)_{I}\right) x \neq\right.\) None \()=\)
ListMem x (strips-problem.variables-of \((\varphi \Psi)\) )
using is-valid-problem-sas-plus-then-strips-transformation-too-iv[OF assms].
moreover have \(\forall x .\left((\varphi \Psi)_{G}\right) x \neq\) None \(\longrightarrow\)
ListMem x (strips-problem.variables-of \((\varphi \Psi)\) )
using is-valid-problem-sas-plus-then-strips-transformation-too-v[OF assms].
ultimately show ?thesis
using is-valid-problem-strips-def
unfolding STRIPS-Representation.is-valid-problem-strips-def
by fastforce
qed
lemma set-filter-all-possible-assignments-true-is:
assumes is-valid-problem-sas-plus \(\Psi\)
shows set (filter \((\lambda(v, a) . s(v, a)=\) Some True \()\)
(all-possible-assignments-for \(\Psi)\) )
\(=\left(\bigcup v \in \operatorname{set}\left((\Psi)_{\mathcal{V}}\right)\right.\). Pair \(v '\left\{a \in \mathcal{R}_{+} \Psi v . s(v, a)=\right.\) Some True \(\left.\}\right)\)
proof -
let ?vs \(=\) sas-plus-problem.variables-of \(\Psi\)
and \(? P=(\lambda(v, a) . s(v, a)=\) Some True \()\)
let ?l \(=\) filter ? P (all-possible-assignments-for \(\Psi)\)
have set ?l = set (concat (map (filter ?P) (map (possible-assignments-for \(\Psi\) ) ? \(v s)\) ))
unfolding all-possible-assignments-for-def
filter-concat[of? ? map (possible-assignments-for \(\Psi\) ) (sas-plus-problem.variables-of
\(\Psi)\)
by \(\operatorname{simp}\)
```

    also have ... = set (concat (map ( }\lambdav.\mathrm{ filter ?P (possible-assignments-for }\Psiv)
    ?vs))
unfolding map-map comp-apply
by blast
also have ... = set (concat (map (\lambdav. map (Pair v)
(filter (?P \circ Pair v)(the (range-of \Psi v)))) ?vs))
unfolding possible-assignments-for-def filter-map
by blast
also have ... = set (concat (map (\lambdav. map (Pair v) (filter (\lambdaa.s (v,a) = Some
True)
(the (range-of \Psi v)))) ?vs))
unfolding comp-apply
by fast
also have ... = \bigcup(set'((\lambdav. map (Pair v) (filter (\lambdaa.s (v,a)=Some True)
(the (range-of \Psi v))))' set ?vs))
unfolding set-concat set-map..
also have ... = ( \bigcupv veset ?vs. Pair v'set (filter (\lambdaa.s (v,a)=Some True)
(the (range-of \Psi v))))
unfolding image-comp[of set] comp-apply set-map..
also have ... = ( \bigcupv vest ?vs. Pair v
'{ { a set (the (range-of \Psi v)).s(v,a)=Some True })
unfolding set-filter..
finally show ?thesis
using set-the-range-of-is-range-of-sas-plus-if[OF assms(1)]
by auto
qed
lemma strips-state-to-state-dom-is:
assumes is-valid-problem-sas-plus \Psi
shows dom( }\mp@subsup{\varphi}{S}{-1}\Psis
=(\bigcupv\in\operatorname{set}((\Psi)\mp@subsup{\mathcal{V}}{+}{\prime}).
{v|a.a\in(\mathcal{R}+\Psiv)^s(v,a)=Some True })
proof -
let ?vs= variables-of }
and ?s' = \varphi }\mp@subsup{S}{S}{-1}\Psi
and ?P}=(\lambda(v,a).s(v,a)=\mathrm{ Some True)
let ?l = filter ?P (all-possible-assignments-for }\Psi
{
have fst' set ?l = fst' ( }\vv\in\mathrm{ set ?vs. Pair v
' {a\in\mathcal{R}}+\Psi\mathrm{ Iv.s (v,a)=Some True })
unfolding set-filter-all-possible-assignments-true-is[OF assms]
by auto
also have ... = (\bigcupv\in set ?vs. fst 'Pair v
'{a\in\mathcal{R}+\Psiv.s(v,a)=Some True })
by blast
also have ... =( \bigcupv\in set ?vs. (\lambdaa.fst (Pair va))'
{a\in\mathcal{R}}+\Psiv.s(v,a)=\mathrm{ Some True })
unfolding image-comp[of fst] comp-apply
by blast

```
```

    finally have fst' set ?l = ( Uv\in set ((\Psi)}\mp@subsup{\mathcal{V}}{+}{})\mathrm{ .
        {v|a.a\in(\mp@subsup{\mathcal{R}}{+}{}\Psiv)\wedges(v,a)=\mathrm{ Some True })}
        unfolding setcompr-eq-image fst-conv
        by simp
    }
    thus ?thesis
    unfolding SAS-Plus-STRIPS.strips-state-to-state-def
        strips-state-to-state-def dom-map-of-conv-image-fst
    by blast
    qed
lemma strips-state-to-state-range-is:
assumes is-valid-problem-sas-plus \Psi
and}v\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{V}}{+}
and a\in\mathcal{R}+\Psiv
and}(v,a)\indom s
and }\forall(v,a)\in\operatorname{dom}\mp@subsup{s}{}{\prime}.\forall(v,\mp@subsup{a}{}{\prime})\in\operatorname{dom}\mp@subsup{s}{}{\prime}.\mp@subsup{s}{}{\prime}(v,a)=Some True^ s'(v, a')
Some True
\longrightarrow ( v , a ) = ( v , a ^ { \prime } )
shows (\varphi\mp@subsup{S}{}{-1}\Psi s}\mp@subsup{s}{}{\prime})v=\mathrm{ Some }a\longleftrightarrow\mathrm{ the (s'(v,a))
proof -
let ?vs = variables-of }
and ?D = range-of }
and ?s}=\mp@subsup{\varphi}{S}{}\mp@subsup{}{}{-1}\Psi\mp@subsup{s}{}{\prime
let ?as = all-possible-assignments-for }
let ?l = filter ( }\lambda(v,a).\mp@subsup{s}{}{\prime}(v,a)=\mathrm{ Some True) ?as
show ?thesis
proof (rule iffI)
assume s-of-v-is-Some-a: ?s v = Some a
{
have (v,a)\in set ?l
using s-of-v-is-Some-a
unfolding SAS-Plus-STRIPS.strips-state-to-state-def
strips-state-to-state-def
using map-of-SomeD
by fast
hence s' (v,a) = Some True
unfolding all-possible-assignments-for-set-is set-filter
by blast
}
thus the (s'(v,a))
by simp
next
assume the-of-s'-of-v-a-is: the (s'}(v,a)
then have }\mp@subsup{s}{}{\prime}\mathrm{ -of-v-a-is-Some-true: }\mp@subsup{s}{}{\prime}(v,a)=Some Tru
using assms(4) domIff
by force
- TODO slow.
moreover {

```
```

        fix }v\mp@subsup{v}{}{\prime}a\mp@subsup{a}{}{\prime
        assume (v,a)\in set ?l and ( v},\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\mathrm{ set ?l
        then have v\not=\mp@subsup{v}{}{\prime}\veea=\mp@subsup{a}{}{\prime}
        using assms(5)
        by fastforce
    }
    moreover {
        have }\forallv\in\mathrm{ set (( }\Psi\mp@subsup{)}{\mathcal{V}++}{})\mathrm{ . sas-plus-problem.range-of }\Psiv\not=Non
            using is-valid-problem-sas-plus-then(1) assms(1)
                range-of-not-empty
            by force
    moreover have set ?l = Set.filter (\lambda(v,a). s'(v,a)=Some True)
            (\bigcupv\in\operatorname{set}((\Psi)\mp@subsup{\mathcal{V}}{+}{\prime}).{(v,a)|a.a\in\mathcal{R}
            using all-possible-assignments-for-set-is calculation
            by force
    ultimately have (v,a)\in set ?l
        using assms(2, 3) s'-of-v-a-is-Some-true
        by simp
    }
    ultimately show ?s v = Some a
        using map-of-constant-assignments-defined-if[of ?l v a]
        unfolding SAS-Plus-STRIPS.strips-state-to-state-def
        strips-state-to-state-def
        by blast
    qed
    qed

```
- NOTE A technical lemma which characterizes the return values for possible assignments \((v, a)\) when used as variables on a state \(s\) which was transformed from.
```

lemma strips-state-to-state-inverse-is- $i$ :
assumes is-valid-problem-sas-plus $\Psi$
and $v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)$
and $s v \neq$ None
and $a \in \mathcal{R}_{+} \Psi v$
shows $\left(\varphi_{S} \Psi s\right)(v, a)=$ Some $($ the $(s v)=a)$
proof -
let ?vs $=$ sas-plus-problem.variables-of $\Psi$
let ? $s^{\prime}=\varphi_{S} \Psi s$
and $? f=\lambda(v, a)$. the $(s v)=a$
and $? l=$ concat (map (possible-assignments-for $\Psi)($ filter $(\lambda v . s v \neq$ None)
?vs))
have $(v, a) \in \operatorname{dom}$ ? $s^{\prime}$
using state-to-strips-state-dom-element-iff [
OF $\operatorname{assms}(1)] \operatorname{assms}(2,3,4)$
by presburger
\{
have $v \in\left\{v \mid v . v \in \operatorname{set}\left((\Psi)_{\mathcal{V}+}\right) \wedge s v \neq\right.$ None $\}$

```
```

    using assms(2, 3)
    by blast
    moreover have }\forallv\in\operatorname{set}((\Psi\mp@subsup{)}{\mp@subsup{\mathcal{V}}{+}{\prime}}{)}.v\in\operatorname{dom}\mathrm{ (sas-plus-problem.range-of }\Psi\mathrm{ )
    using is-valid-problem-sas-plus-dom-sas-plus-problem-range-of[OF assms(1)].
    ```

```

        {(v,a)|a.a\in\mathcal{R}
        unfolding state-to-strips-state-dom-is-i[OF calculation(2)]
        by blast
    ultimately have (v,a)\in set ?l
        using assms(4)
        by blast
    }
moreover have set ?l }\not={
using calculation
by force
- TODO slow.
ultimately show ?thesis
unfolding SAS-Plus-STRIPS.state-to-strips-state-def
state-to-strips-state-def
using map-of-from-function-graph-is-some-if[of ?l (v, a)?f]
unfolding split-def
by fastforce
qed

```
- NOTE Show that the transformed strips state is consistent for pairs of assignments \((v, a)\) and \(\left(v, a^{\prime}\right)\) in the same variable domain.
```

corollary strips-state-to-state-inverse-is-ii:
assumes is-valid-problem-sas-plus $\Psi$
and $v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)$
and $s v=$ Some $a$
and $a \in \mathcal{R}_{+} \Psi v$
and $a^{\prime} \in \mathcal{R}_{+} \Psi v$
and $a^{\prime} \neq a$
shows $\left(\varphi_{S} \Psi s\right)\left(v, a^{\prime}\right)=$ Some False
proof -
have $s v \neq$ None
using assms(3)
by $\operatorname{simp}$
moreover have the ( $s v$ ) $\neq a^{\prime}$
using $\operatorname{assms}(3,6)$
by $\operatorname{simp}$
ultimately show ?thesis
using strips-state-to-state-inverse-is-i $[$ OF $\operatorname{assms}(1,2)-\operatorname{assms}(5)]$
by force
qed

```
- NOTE Follows from the corollary above by contraposition.
```

corollary strips-state-to-state-inverse-is-iii:
assumes is-valid-problem-sas-plus $\Psi$
and $v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)$
and $s v=$ Some $a$
and $a \in \mathcal{R}_{+} \Psi v$
and $a^{\prime} \in \mathcal{R}_{+} \Psi v$
and $\left(\varphi_{S} \Psi s\right)(v, a)=$ Some True
and $\left(\varphi_{S} \Psi s\right)\left(v, a^{\prime}\right)=$ Some True
shows $a=a^{\prime}$
proof -
have $s v \neq$ None
using assms(3)
by blast
thus ?thesis
using strips-state-to-state-inverse-is-i $[O F \operatorname{assms}(1,2)] \operatorname{assms}(4,5,6,7)$
by auto
qed
lemma strips-state-to-state-inverse-is-iv:
assumes is-valid-problem-sas-plus $\Psi$
and $\operatorname{dom} s \subseteq \operatorname{set}\left((\Psi)_{\mathcal{V}+}\right)$
and $v \in \operatorname{set}\left((\Psi)_{\mathcal{V}+}\right)$
and $s v=$ Some $a$
and $a \in \mathcal{R}_{+} \Psi v$
shows $\left(\varphi_{S}{ }^{-1} \Psi\left(\varphi_{S} \Psi s\right)\right) v=$ Some $a$
proof -
let $? v s=$ variables-of $\Psi$
and $? s^{\prime}=\varphi_{S} \Psi s$
let ${ }^{\prime} s^{\prime \prime}=\varphi_{S^{-1}} \Psi$ ? $s^{\prime}$
let ? $P=\lambda(v, a)$. ? $s^{\prime}(v, a)=$ Some True
let ?as $=$ filter ? $P$ (all-possible-assignments-for $\Psi)$
and ?As $=$ Set.filter ? $P\left(\bigcup v \in \operatorname{set}\left((\Psi)_{\mathcal{V}+}\right)\right.$.
$\left.\left\{(v, a) \mid a . a \in \mathcal{R}_{+} \Psi v\right\}\right)$
\{
have $\forall v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)$. range-of $\Psi v \neq$ None
using sublocale-sas-plus-finite-domain-representation-ii(1)[OF assms(1)]
range-of-not-empty
by force
hence set ?as $=$ ? As
unfolding set-filter
using all-possible-assignments-for-set-is
by force
$\}$ note $n b=$ this
moreover \{
\{
fix $v v^{\prime} a a^{\prime}$

```
```

    assume (v,a)\in set ?as
    and ( }\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\mathrm{ set ?as
    then have (v,a)\in?As and ( }\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in?A
    using nb
    by blast+
    then have v-in-set-vs:v\in set ?vs and v'-in-set-vs: v'\in set ?vs
    and a-in-range-of-v:}a\in\mp@subsup{\mathcal{R}}{+}{}\Psi
    and a'-in-range-of-v: a'}\mp@subsup{a}{}{\prime}\in\mp@subsup{\mathcal{R}}{+}{}\Psi\mp@subsup{v}{}{\prime
    and s'-of-v-a-is: ? s' (v,a) = Some True and s'-of-v'- 'a'-is: ?s' (v', a')=
    Some True
by fastforce+
then have (v,a)\indom? ? '
by blast
then have s-of-v-is-Some-a: s v=Some a
using state-to-strips-state-dom-element-iff [OF assms(1)]
state-to-strips-state-range-is[OF assms(1)] s'-of-v-a-is
by auto
have v\not=\mp@subsup{v}{}{\prime}\vee a= a
proof (rule ccontr)
assume}\neg(v\not=\mp@subsup{v}{}{\prime}\veea=\mp@subsup{a}{}{\prime}
then have v=\mp@subsup{v}{}{\prime}}\mathrm{ and }a\not=\mp@subsup{a}{}{\prime
by simp+
thus False
using a'-in-range-of-v a-in-range-of-v assms(1) v'-in-set-vs s'-of-v'-a'-is
s'-of-v-a-is s-of-v-is-Some-a strips-state-to-state-inverse-is-iii
by force
qed
}
moreover {
have s v\not= None
using assms(4)
by simp
then have ?s'}(v,a)=\mathrm{ Some True
using strips-state-to-state-inverse-is-i[OF assms(1, 3) - assms(5)]
assms(4)
by simp
hence (v,a) \in set ?as
using all-possible-assignments-for-set-is assms(3,5) nb
by simp
}
ultimately have map-of ?as v = Some a
using map-of-constant-assignments-defined-if[of ?as v a]
by blast
}

- TODO slow.
thus ?thesis
unfolding SAS-Plus-STRIPS.strips-state-to-state-def
strips-state-to-state-def all-possible-assignments-for-def

```
```

    by simp
    qed

```
- Show that that \(\varphi_{S}{ }^{-1} \Psi\) is the inverse of \(\varphi_{S} \Psi\). The additional constraints \(\operatorname{dom} s=\operatorname{set}\left(\Psi_{\mathcal{V}_{+}}\right)\)and \(\forall v \in \operatorname{dom} s\). the \((s v) \in \mathcal{R}_{+} \Psi v\) are needed because the transformation functions only take into account variables and domains declared in the problem description. They also sufficiently characterize a state that was transformed from SAS+ to STRIPS.
lemma strips-state-to-state-inverse-is:
assumes is-valid-problem-sas-plus \(\Psi\)
and \(\operatorname{dom} s \subseteq \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
and \(\forall v \in\) dom s. the \((s v) \in \mathcal{R}_{+} \Psi v\)
shows \(s=\left(\varphi_{S}{ }^{-1} \Psi\left(\varphi_{S} \Psi s\right)\right)\)
proof -
let ?vs \(=\) variables-of \(\Psi\)
and \(? D=\) range-of \(\Psi\)
let \(?^{\prime} s^{\prime}=\varphi_{S} \Psi s\)
let ? \(s^{\prime \prime}=\varphi_{S}{ }^{-1} \Psi ? s^{\prime}\)
- NOTE Show the thesis by proving that \(s\) and \(? s^{\prime}\) are mutual submaps.
\{
fix \(v\)
assume \(v\)-in-dom-s: \(v \in \operatorname{dom} s\)
then have \(v\)-in-set-vs: \(v \in\) set ?vs
using assms(2)
by auto
then obtain \(a\)
where the-s-v-is-a: s \(v=\) Some \(a\)
and \(a\)-in-dom-v: \(a \in \mathcal{R}_{+} \Psi v\)
using \(\operatorname{assms}(2,3) v\)-in-dom-s
by force
moreover have ? \(s^{\prime \prime} v=\) Some \(a\)
using strips-state-to-state-inverse-is-iv[OF \(\operatorname{assms}(1,2)] v\)-in-set-vs the-s-v-is-a a-in-dom-v
by force
ultimately have \(s v=? s^{\prime \prime} v\)
by argo
\} note \(n b=\) this
moreover \{
fix \(v\)
assume \(v \in \operatorname{dom} ? s^{\prime \prime}\)
then obtain \(a\)
where \(a \in \mathcal{R}_{+} \Psi v\)
and \(? s^{\prime}(v, a)=\) Some True
using strips-state-to-state-dom-is[OF assms(1)]
by blast
then have \((v, a) \in d o m ? s^{\prime}\)
by blast
```

    then have s v\not= None
    using state-to-strips-state-dom-is[OF assms(1)]
    by simp
    then obtain a where s v=Some a
    by blast
    hence ?s" v=sv
    using nb
    by fastforce
    }

- TODO slow.
ultimately show ?thesis
using map-le-antisym[of s ? ''|] map-le-def
unfolding strips-state-to-state-def
state-to-strips-state-def
by blast
qed

```
- An important lemma which shows that the submap relation does not change if we transform the states on either side from SAS+ to STRIPS.
lemma state-to-strips-state-map-le-iff:
assumes is-valid-problem-sas-plus \(\Psi\)
and \(\operatorname{dom} s \subseteq \operatorname{set}\left((\Psi) \mathcal{V}_{+}\right)\)
and \(\forall v \in\) dom s. the \((s v) \in \mathcal{R}_{+} \Psi v\)
shows \(s \subseteq_{m} t \longleftrightarrow\left(\varphi_{S} \Psi s\right) \subseteq_{m}\left(\varphi_{S} \Psi t\right)\)
proof -
let ?vs \(=\) variables-of \(\Psi\)
and \(? D=\) range-of \(\Psi\)
and \(? s^{\prime}=\varphi_{S} \Psi s\)
and \(? t^{\prime}=\varphi_{S} \Psi t\)
show ?thesis
proof (rule iffi)
assume s-map-le-t: \(s \subseteq_{m} t\)
\{
fix \(v a\)
assume \((v, a) \in \operatorname{dom} ? s^{\prime}\)
moreover have \(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)and \(s v \neq\) None and \(a \in \mathcal{R}_{+} \Psi v\)
using state-to-strips-state-dom-is[OF assms(1)] calculation
by blast+
moreover have ? \(s^{\prime}(v, a)=\) Some (the \(\left.(s v)=a\right)\)
using state-to-strips-state-range-is[OF assms(1)] calculation(1)
by meson
moreover have \(v \in\) dom \(s\) using calculation(3)
by auto
moreover have \(s v=t v\)
using s-map-le-t calculation(6)
unfolding map-le-def
by blast
moreover have \(t v \neq\) None
```

        using calculation(3, 7)
        by argo
    moreover have (v,a)\indom ? t'
        using state-to-strips-state-dom-is[OF assms(1)] calculation(2, 4, 8)
        by blast
    moreover have ? t' (v,a)=Some (the (tv)=a)
        using state-to-strips-state-range-is[OF assms(1)] calculation(9)
        by simp
    ultimately have ? s}\mp@subsup{s}{}{\prime}(v,a)=?\mp@subsup{t}{}{\prime}(v,a
    by presburger
    }
thus ?s' }\mp@subsup{\subseteq}{m}{}\mathrm{ ? ?t'
unfolding map-le-def
by fast
next
assume s'-map-le-t': ?s' }\mp@subsup{\subseteq}{m}{}
{
fix v
assume v-in-dom-s: v\indom s
moreover obtain a where the-of-s-of-v-is-a: the (s v)=a
by blast
moreover have v-in-vs:v\in set ((\Psi)\mathcal{V}+)
and s-of-v-is-not-None: s v}\not=\mathrm{ None
and a-in-range-of-v: a\in\mathcal{R}
using assms(2, 3) v-in-dom-s calculation
by blast+
moreover have (v,a)\indom ?s'
using state-to-strips-state-dom-is[OF assms(1)]
calculation(3, 4, 5)
by simp
moreover have ?s' (v,a) =? ?t'}(v,a
using s'-map-le-t' calculation
unfolding map-le-def
by blast
moreover have (v,a)\indom ? t'
using calculation
unfolding domIff
by argo
moreover have ?s' (v,a)=Some (the (s v)=a)
and ?t' (v,a) = Some (the ( }tv)=a
using state-to-strips-state-range-is[OF assms(1)] calculation
by fast+
moreover have s v=Some a
using calculation(2, 4)
by force
moreover have ? s' (v,a) = Some True
using calculation(9, 11)
by fastforce
moreover have ?t' (v,a) = Some True

```
```

            using calculation(7, 12)
            by argo
            moreover have the (tv)=a
            using calculation(10, 13) try0
            by force
            moreover {
            have}v\in\operatorname{dom}
                using state-to-strips-state-dom-element-iff[OF assms(1)]
                    calculation(8)
                by auto
            hence tv= Some a
                    using calculation(14)
            by force
        }
        ultimately have s v=tv
            by argo
    }
    thus }s\mp@subsup{\subseteq}{m}{}
        unfolding map-le-def
        by simp
    qed
    qed

```
- We also show that \(\varphi_{O}{ }^{-1} \Pi\) is the inverse of \(\varphi_{O} \Psi\). Note that this proof is completely mechanical since both the precondition and effect lists are simply being copied when transforming from SAS+ to STRIPS and when transforming back from STRIPS to SAS+.
lemma sas-plus-operator-inverse-is:
assumes is-valid-problem-sas-plus \(\Psi\)
and \(o p \in \operatorname{set}\left((\Psi)_{\mathcal{O}+}\right)\)
shows \(\left(\varphi_{O}{ }^{-1} \Psi\left(\varphi_{O} \Psi o p\right)\right)=o p\)
proof -
let ? \(o p=\varphi_{O}{ }^{-1} \Psi\left(\varphi_{O} \Psi o p\right)\)
have precondition-of ? op \(=\) precondition-of op
unfolding SAS-Plus-STRIPS.strips-op-to-sasp-def
strips-op-to-sasp-def
SAS-Plus-STRIPS.sasp-op-to-strips-def
sasp-op-to-strips-def
by fastforce
moreover have effect-of ?op \(=\) effect-of op
unfolding SAS-Plus-STRIPS.strips-op-to-sasp-def
strips-op-to-sasp-def
SAS-Plus-STRIPS.sasp-op-to-strips-def
sasp-op-to-strips-def
by force
ultimately show ?thesis
by \(\operatorname{simp}\)

\section*{qed}
- Note that we have to make the assumption that \(o p^{\prime}\) is a member of the operator set of the induced STRIPS problem \(\varphi \Psi\). This implies that \(o p^{\prime}\) was transformed from an \(o p \in\) operators-of \(\Psi\). If we don't make this assumption, then multiple STRIPS operators of the form 1 precondition-of \(=[]\), add-effects-of \(=[]\), delete-effects-of \(=[(v, a), \ldots] D\) correspond to one SAS+ operator (since the delete effects are being discarded in the transformation function).
lemma strips-operator-inverse-is:
assumes is-valid-problem-sas-plus \(\Psi\)
and \(o p^{\prime} \in \operatorname{set}\left((\varphi \Psi)_{\mathcal{O}}\right)\)
shows \(\left(\varphi_{O} \Psi\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\right)=o p^{\prime}\)
proof -
let \(? \Pi=\varphi \Psi\)
obtain \(o p\) where \(o p \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)and \(o p^{\prime}=\varphi_{O} \Psi o p\)
using assms
by auto
moreover have \(\varphi_{O}{ }^{-1} \Psi o p^{\prime}=o p\)
using sas-plus-operator-inverse-is[OF assms(1) calculation(1)] calculation(2) by blast
ultimately show ?thesis
by argo
qed
lemma sas-plus-equivalent-to-strips-i-a-I:
assumes is-valid-problem-sas-plus \(\Psi\)
and set ops \({ }^{\prime} \subseteq \operatorname{set}\left((\varphi \Psi)_{\mathcal{O}}\right)\)
and STRIPS-Semantics.are-all-operators-applicable \(\left(\varphi_{S} \Psi\right.\) s) ops \({ }^{\prime}\)
and \(o p \in \operatorname{set}\left[\varphi_{O}{ }^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow o p s\right]\)
shows map-of (precondition-of op) \(\subseteq_{m}\left(\varphi_{S}{ }^{-1} \Psi\left(\varphi_{S} \Psi s\right)\right)\)
proof -
let \(? \Pi=\varphi \Psi\)
and \(? s^{\prime}=\varphi_{S} \Psi s\)
let ? \(s=\varphi_{S}{ }^{-1} \Psi\) ? \(s^{\prime}\)
and \(? D=\) range-of \(\Psi\)
and ?ops \(=\left[\varphi_{O}{ }^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow o p s^{\prime}\right]\)
and ?pre \(=\) precondition-of op
have \(n b_{1}: \forall(v, a) \in \operatorname{dom}\) ? \(s^{\prime}\).
\(\forall\left(v, a^{\prime}\right) \in \operatorname{dom}\) ? \(s^{\prime}\).
\(? s^{\prime}(v, a)=\) Some True \(\wedge ? s^{\prime}\left(v, a^{\prime}\right)=\) Some True \(\longrightarrow(v, a)=\left(v, a^{\prime}\right)\)
using state-to-strips-state-effect-consistent[OF assms(1)]
by blast
\{
fix \(o p^{\prime}\)
assume \(o p^{\prime} \in\) set ops \({ }^{\prime}\)
moreover have \(o p^{\prime} \in \operatorname{set}\left((? \Pi)_{\mathcal{O}}\right)\)
using assms(2) calculation
by blast
ultimately have \(\exists o p \in \operatorname{set}\left((\Psi)_{\mathcal{O}+}\right)\). \(o p^{\prime}=\left(\varphi_{O} \Psi o p\right)\)
by auto
\(\}\) note \(n b_{2}=\) this
\{
fix \(o p\)
assume \(o p \in\) set ?ops
then obtain \(o p^{\prime}\) where \(o p^{\prime} \in\) set \(o p s^{\prime}\) and \(o p=\varphi_{O}{ }^{-1} \Psi o p^{\prime}\)
using assms(4)
by auto
moreover obtain \(o p^{\prime \prime}\) where \(o p^{\prime \prime} \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)and \(o p^{\prime}=\varphi_{O} \Psi o p^{\prime \prime}\)
using \(n b_{2}\) calculation(1)
by blast
moreover have \(o p=o p^{\prime \prime}\)
using sas-plus-operator-inverse-is[OF assms(1) calculation(3)] calculation(2,
4)
by blast
ultimately have \(o p \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)
by blast
\(\}\) note \(n b_{3}=t h i s\)
\{
fix \(o p\) \(v a\)
assume op \(\in\) set ?ops
and \(v\)-a-in-precondition-of-op \({ }^{\prime}:(v, a) \in \operatorname{set}\) (precondition-of op)
moreover obtain \(o p^{\prime}\) where \(o p^{\prime} \in\) set \(o p s^{\prime}\) and \(o p=\varphi_{O}{ }^{-1} \Psi o p^{\prime}\)
using calculation(1)
by auto
moreover have strips-operator.precondition-of \(o p^{\prime}=\) precondition-of op
using calculation(4)
unfolding SAS-Plus-STRIPS.strips-op-to-sasp-def
strips-op-to-sasp-def
by \(\operatorname{simp}\)
ultimately have \(\exists o p^{\prime} \in\) set ops \({ }^{\prime} . o p=\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\)
\(\wedge(v, a) \in\) set (strips-operator.precondition-of op')
by metis
\(\}\) note \(n b_{4}=\) this
\{
fix \(o p^{\prime} v a\)
assume \(o p^{\prime} \in\) set ops \({ }^{\prime}\)
and \(v\)-a-in-precondition-of-op \(:(v, a) \in\) set (strips-operator.precondition-of
\(o p^{\prime}\) )
moreover have \(s^{\prime}\)-of-v-a-is-Some-True: ? \(s^{\prime}(v, a)=\) Some True
using assms(3) calculation(1, 2)
unfolding are-all-operators-applicable-set
by blast
moreover \{
obtain \(o p\) where \(o p \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)and \(o p^{\prime}=\varphi_{O} \Psi o p\)
using \(n b_{2}\) calculation(1)
by blast
```

    moreover have strips-operator.precondition-of op' = precondition-of op
        using calculation(2)
        unfolding SAS-Plus-STRIPS.sasp-op-to-strips-def
        sasp-op-to-strips-def
    by simp
    moreover have (v,a)\in set (precondition-of op)
        using v-a-in-precondition-of-op' calculation(3)
        by argo
    moreover have is-valid-operator-sas-plus \Psi op
        using is-valid-problem-sas-plus-then(2) assms(1) calculation(1)
        unfolding is-valid-operator-sas-plus-def
        by auto
    moreover have }v\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{V}+}{+})\mathrm{ and }a\in\mp@subsup{\mathcal{R}}{+}{}\Psi
        using is-valid-operator-sas-plus-then(1,2) calculation(4, 5)
        unfolding is-valid-operator-sas-plus-def
        by fastforce+
    moreover have v\in dom?s
        using strips-state-to-state-dom-is[OF assms(1), of ?s}
        s'-of-v-a-is-Some-True calculation(6, 7)
        by blast
    moreover have (v,a)\indom?s'
        using s'-of-v-a-is-Some-True domIff
        by blast
    ultimately have ?s v}=\mathrm{ Some a
    using strips-state-to-state-range-is[OF assms(1) --nb [ ]
        s'-of-v-a-is-Some-True
    by simp
    }
hence ?s v = Some a.
} note nb b}=thi
{
fix v
assume v\indom (map-of ?pre)
then obtain a where map-of ?pre v=Some a
by fast
moreover have (v,a)\in set ?pre
using map-of-SomeD calculation
by fast
moreover {
have op }\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O+}}{}
using assms(4) nb
by blast
then have is-valid-operator-sas-plus \Psi op
using is-valid-problem-sas-plus-then(2) assms(1)
unfolding is-valid-operator-sas-plus-def
by auto
hence }\forall(v,a)\in\mathrm{ set ?pre. }\forall(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\mathrm{ set ?pre. }v\not=\mp@subsup{v}{}{\prime}\veea=\mp@subsup{a}{}{\prime
using is-valid-operator-sas-plus-then(5)
unfolding is-valid-operator-sas-plus-def

```
```

        by fast
    }
    moreover have map-of ?pre v = Some a
        using map-of-constant-assignments-defined-if[of ?pre] calculation(2, 3)
        by blast
    moreover obtain op' where o\mp@subsup{p}{}{\prime}\in set ops'
        and (v,a)\in set (strips-operator.precondition-of op')
        using n\mp@subsup{b}{4}{}[OF assms(4) calculation(2)]
        by blast
    moreover have ?s v = Some a
        using n\mp@subsup{b}{5}{}}\mathrm{ calculation(5, 6)
        by fast
    ultimately have map-of ?pre v=?sv
    by argo
    }
thus ?thesis
unfolding map-le-def
by blast
qed
lemma to-sas-plus-list-of-transformed-sas-plus-problem-operators-structure:
assumes is-valid-problem-sas-plus \Psi
and set ops'}\subseteq\operatorname{set}((\varphi\Psi\mp@subsup{)}{\mathcal{O}}{}
and op \in set [ }\mp@subsup{\varphi}{O}{-1}\Psi \Psio\mp@subsup{p}{}{\prime}.o\mp@subsup{p}{}{\prime}\leftarrowops'
shows op \in set }((\Psi\mp@subsup{)}{\mathcal{O+}}{})\wedge(\existso\mp@subsup{p}{}{\prime}\in\mathrm{ set ops'.op'}=\mp@subsup{\varphi}{O}{\prime}\Psiop
proof -
let ?\Pi=\varphi\Psi
obtain o\mp@subsup{p}{}{\prime}\mathrm{ where }o\mp@subsup{p}{}{\prime}\in\mathrm{ set ops' and op = 的 -1 }\Psi o\mp@subsup{p}{}{\prime}
using assms(3)
by auto
moreover have op'\in set ((?\Pi)
using assms(2) calculation(1)
by blast
moreover obtain o\mp@subsup{p}{}{\prime\prime}}\mathrm{ where }o\mp@subsup{p}{}{\prime\prime}\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O+}}{})\mathrm{ and }o\mp@subsup{p}{}{\prime}=\mp@subsup{\varphi}{O}{}\Psi\mp@code{*p
using calculation(3)
by auto
moreover have op =op"
using sas-plus-operator-inverse-is[OF assms(1) calculation(4)] calculation(2,
5)
by presburger
ultimately show ?thesis
by blast
qed
lemma sas-plus-equivalent-to-strips-i-a-II:
fixes \Psi :: ('variable, 'domain) sas-plus-problem
fixes s:: ('variable, 'domain) state
assumes is-valid-problem-sas-plus \Psi

```
and set ops \({ }^{\prime} \subseteq \operatorname{set}\left((\varphi \Psi)_{\mathcal{O}}\right)\)
and STRIPS-Semantics.are-all-operators-applicable \(\left(\varphi_{s} \Psi\right.\) s) ops \({ }^{\prime}\)
\(\wedge\) STRIPS-Semantics.are-all-operator-effects-consistent ops \({ }^{\prime}\)
shows are-all-operator-effects-consistent \(\left[\varphi_{O}{ }^{-1} \Psi\right.\) op \({ }^{\prime}\). op \(\left.{ }^{\prime} \leftarrow o p s^{\prime}\right]\)
proof -
let ? \(s^{\prime}=\varphi_{S} \Psi s\)
let ?s \(=\varphi_{S^{-1}} \Psi\) ? \(s^{\prime}\)
and ?ops \(=\left[\varphi_{O}{ }^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow o p s^{\prime}\right]\)
and ? \(\Pi=\varphi \Psi\)
have \(n b: \forall(v, a) \in \operatorname{dom} ? s^{\prime}\).
\(\forall\left(v, a^{\prime}\right) \in \operatorname{dom}\) ? \(s^{\prime}\).
\(? s^{\prime}(v, a)=\) Some True \(\wedge ? s^{\prime}\left(v, a^{\prime}\right)=\) Some True \(\longrightarrow(v, a)=\left(v, a^{\prime}\right)\)
using state-to-strips-state-effect-consistent[OF assms(1)]
by blast
\{
fix \(o p_{1}{ }^{\prime} o p_{2}{ }^{\prime}\)
assume \(o p_{1}{ }^{\prime} \in\) set ops \({ }^{\prime}\) and \(o p_{2}{ }^{\prime} \in\) set ops \({ }^{\prime}\)
hence STRIPS-Semantics.are-operator-effects-consistent \(o p_{1}{ }^{\prime} o p_{2}{ }^{\prime}\)
using assms(3)
unfolding STRIPS-Semantics.are-all-operator-effects-consistent-def list-all-iff
by blast
\(\}\) note \(n b_{1}=t h i s\)
\{
fix \(o p_{1} o p_{1}{ }^{\prime} o p_{2} o p_{2}{ }^{\prime}\)
assume op \(p_{1}\)-in-ops: op \(p_{1} \in\) set ?ops
and \(o p_{1}{ }^{\prime}-i n\)-ops \({ }^{\prime}: o p_{1}{ }^{\prime} \in\) set ops \({ }^{\prime}\)
and \(o p_{1}{ }^{\prime}-i s: o p_{1}{ }^{\prime}=\varphi_{O} \Psi o p_{1}\)
and is-valid-op \(p_{1}\) : is-valid-operator-sas-plus \(\Psi\) op \({ }_{1}\)
and \(o p_{2}\)-in-ops: \(o p_{2} \in\) set ?ops
and \(o p_{2}{ }^{\prime}\)-in-ops \({ }^{\prime}: o p_{2}{ }^{\prime} \in\) set ops \({ }^{\prime}\)
and \(o p_{2}{ }^{\prime}\)-is: \(o p_{2}{ }^{\prime}=\varphi_{O} \Psi o p_{2}\)
and is-valid-op \(p_{2}\) : is-valid-operator-sas-plus \(\Psi\) op 2
have \(\forall(v, a) \in \operatorname{set}\left(\right.\) add-effects-of op \(\left.{ }_{1}{ }^{\prime}\right) . \forall\left(v^{\prime}, a^{\prime}\right) \in \operatorname{set}\left(\right.\) add-effects-of op \(\left.{ }_{2}{ }^{\prime}\right)\). \(v \neq v^{\prime} \vee a=a^{\prime}\)
proof (rule ccontr)
assume \(\neg\left(\forall(v, a) \in \operatorname{set}\left(a d d\right.\right.\)-effects-of op \(\left.{ }_{1}{ }^{\prime}\right) . \forall\left(v^{\prime}, a^{\prime}\right) \in\) set (add-effects-of
\(\left.o p_{2}{ }^{\prime}\right)\).
\(\left.v \neq v^{\prime} \vee a=a^{\prime}\right)\)
then obtain \(v v^{\prime} a a^{\prime}\) where \((v, a) \in \operatorname{set}\left(a d d\right.\)-effects-of \(\left.o p_{1}{ }^{\prime}\right)\)
and \(\left(v^{\prime}, a^{\prime}\right) \in \operatorname{set}\left(a d d\right.\)-effects-of \(\left.o p_{2}{ }^{\prime}\right)\)
and \(v=v^{\prime}\)
and \(a \neq a^{\prime}\)
by blast
- TODO slow.
moreover have \((v, a) \in \operatorname{set}\) (effect-of op \(p_{1}\) ) using \(o p_{1}{ }^{\prime}\)-is \(o p_{2}{ }^{\prime}\)-is calculation (1, 2) unfolding SAS-Plus-STRIPS.sasp-op-to-strips-def sasp-op-to-strips-def
```

        by force
        moreover {
            have ( }\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\operatorname{set}(\mathrm{ effect-of op 2}
            using op 2'-is calculation(2)
            unfolding SAS-Plus-STRIPS.sasp-op-to-strips-def
                sasp-op-to-strips-def
            by force
        hence }\mp@subsup{a}{}{\prime}\in\mp@subsup{\mathcal{R}}{+}{}\Psi
            using is-valid-operator-sas-plus-then is-valid-op 2 calculation(3)
            by fastforce
        }
        moreover have (v, a') \in set (delete-effects-of op ( }\mp@subsup{}{}{\prime}\mathrm{ )
        using sasp-op-to-strips-set-delete-effects-is
            op}\mp@subsup{}{1}{\prime}-\mathrm{ -is is-valid-op ( calculation(3, 4, 5, 6)
        by blast
        moreover have }\neg\mathrm{ STRIPS-Semantics.are-operator-effects-consistent op 1''
    op ''
unfolding STRIPS-Semantics.are-operator-effects-consistent-def list-ex-iff
using calculation(2, 3, 7)
by meson
ultimately show False
using assms(3) op ''-in-ops' op ''}\mp@subsup{}{}{\prime}\mathrm{ -in-ops'
unfolding STRIPS-Semantics.are-all-operator-effects-consistent-def list-all-iff
by blast
qed
} note nb
{
fix }o\mp@subsup{p}{1}{}O\mp@subsup{p}{2}{
assume op -in-ops:op
moreover have op -in-operators-of- }\Psi:o\mp@subsup{p}{1}{}\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O+}}{}
and op 2-in-operators-of-\Psi:op 2 }\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O}+}{}
using to-sas-plus-list-of-transformed-sas-plus-problem-operators-structure[OF
assms(1, 2)] calculation
by blast+
moreover have is-valid-operator-op (1: is-valid-operator-sas-plus \Psi op
and is-valid-operator-op 2: is-valid-operator-sas-plus \Psi op
using is-valid-problem-sas-plus-then(2) op (-in-operators-of-\Psi op 2-in-operators-of-\Psi
assms(1)
unfolding is-valid-operator-sas-plus-def
by auto+
moreover obtain o\mp@subsup{p}{1}{\prime}}o\mp@subsup{p}{2}{\prime
where o\mp@subsup{p}{1}{}-in-ops':o\mp@subsup{p}{1}{\prime}}\mp@subsup{}{}{\prime}\in\mathrm{ set ops'
and o\mp@subsup{p}{1}{}-is:o\mp@subsup{p}{1}{\prime}}\mp@subsup{}{}{\prime}=\mp@subsup{\varphi}{O}{}\Psio\mp@subsup{p}{1}{
and op -in-ops':o\mp@subsup{p}{2}{\prime}}\mp@subsup{}{}{\prime}\in\mathrm{ set ops'
and o\mp@subsup{p}{2}{}-is:o\mp@subsup{p}{2}{\prime}}\mp@subsup{}{}{\prime}=\mp@subsup{\varphi}{O}{}\Psio\mp@subsup{p}{2}{
using to-sas-plus-list-of-transformed-sas-plus-problem-operators-structure[OF
assms(1, 2)] op -in-ops op2-in-ops
by blast

```
- TODO slow.
ultimately have \(\forall(v, a) \in \operatorname{set}\left(a d d\right.\)-effects-of \(\left.o p_{1}{ }^{\prime}\right) . \forall\left(v^{\prime}, a^{\prime}\right) \in\) set (add-effects-of \(o p_{2}{ }^{\prime}\) ).
\[
v \neq v^{\prime} \vee a=a^{\prime}
\]
using \(n b_{3}\)
by auto
hence are-operator-effects-consistent op \(p_{1} o p_{2}\)
using \(o p_{1}\)-is \(o p_{2}\)-is
unfolding are-operator-effects-consistent-def
sasp-op-to-strips-def
SAS-Plus-STRIPS.sasp-op-to-strips-def
list-all-iff Let-def
by simp
\}
thus ?thesis
unfolding are-all-operator-effects-consistent-def list-all-iff
by fast
qed
- A technical lemmas used in sas-plus-equivalent-to-strips-i-a showing that the execution precondition is linear w.r.t. to STRIPS transformation to SAS+.
The second premise states that the given STRIPS state corresponds to a consistent SAS+ state (i.e. no two assignments of the same variable to different values exist).
lemma sas-plus-equivalent-to-strips-i-a-IV:
assumes is-valid-problem-sas-plus \(\Psi\)
and set ops \({ }^{\prime} \subseteq \operatorname{set}\left((\varphi \Psi)_{\mathcal{O}}\right)\)
and STRIPS-Semantics.are-all-operators-applicable \(\left(\varphi_{S} \Psi\right.\) s) ops \({ }^{\prime}\)
\(\wedge\) STRIPS-Semantics.are-all-operator-effects-consistent ops'
shows are-all-operators-applicable-in \(\left(\varphi_{S}{ }^{-1} \Psi\left(\varphi_{S} \Psi s\right)\right)\left[\varphi_{O}^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow\right.\) ops \(] \wedge\)
are-all-operator-effects-consistent \(\left[\varphi_{O}{ }^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow o p s{ }^{\prime}\right]\)
proof -
let \(? \Pi=\varphi \Psi\)
and \(? s^{\prime}=\varphi_{S} \Psi s\)
let \(?\) vs \(^{\prime}=\) strips-problem.variables-of ? \(\Pi\)
and \({ }^{\prime}\) ops \({ }^{\prime}=\) strips-problem.operators-of ? \(\Pi\)
and ?vs \(=\) variables-of \(\Psi\)
and \(? D=\) range-of \(\Psi\)
and \(? s=\varphi_{S}{ }^{-1} \Psi\) ? \(s^{\prime}\)
and ?ops \(=\left[\varphi_{O}{ }^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow o p s^{\prime}\right]\)
have \(n b: \forall(v, a) \in d o m ? s^{\prime}\).
\(\forall\left(v, a^{\prime}\right) \in \operatorname{dom}\left(\varphi_{S} \Psi s\right)\).
\(? s^{\prime}(v, a)=\) Some True \(\wedge\) ? \(s^{\prime}\left(v, a^{\prime}\right)=\) Some True \(\longrightarrow(v, a)=\left(v, a^{\prime}\right)\)
using state-to-strips-state-effect-consistent[OF assms(1)]
by blast
\{
have STRIPS-Semantics.are-all-operators-applicable ?s' ops \({ }^{\prime}\)
using assms(3)
by \(\operatorname{simp}\)
moreover have list-all ( \(\lambda o p\). map-of (precondition-of op) \(\subseteq_{m}\) ?s) ?ops
using sas-plus-equivalent-to-strips-i-a-I[OF assms(1) assms(2)] calculation
unfolding list-all-iff
by blast
moreover have list-all ( \(\lambda\) op. list-all (are-operator-effects-consistent op) ?ops)
?ops
using sas-plus-equivalent-to-strips-i-a-II assms nb
unfolding are-all-operator-effects-consistent-def is-valid-operator-sas-plus-def
list-all-iff
by blast
ultimately have are-all-operators-applicable-in ?s ?ops
unfolding are-all-operators-applicable-in-def is-operator-applicable-in-def list-all-iff
by argo
\}
moreover have are-all-operator-effects-consistent ?ops
using sas-plus-equivalent-to-strips-i-a-II assms nb
by \(\operatorname{simp}\)
ultimately show ?thesis
by \(\operatorname{simp}\)
qed
lemma sas-plus-equivalent-to-strips-i-a-VI:
assumes is-valid-problem-sas-plus \(\Psi\)
and \(\operatorname{dom} s \subseteq \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
and \(\forall v \in\) dom s. the \((s v) \in \mathcal{R}_{+} \Psi v\)
and set ops \({ }^{\prime} \subseteq \operatorname{set}\left((\varphi \Psi)_{\mathcal{O}}\right)\)
and are-all-operators-applicable-in \(s\left[\varphi_{O}{ }^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow o p s\right] \wedge\)
are-all-operator-effects-consistent \(\left[\varphi_{O}{ }^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow o p s^{\prime}\right]\)
shows STRIPS-Semantics.are-all-operators-applicable \(\left(\varphi_{S} \Psi s\right)\) ops \({ }^{\prime}\) proof -
let \(? v s=\) variables-of \(\Psi\)
and \(? D=\) range-of \(\Psi\)
and \(? \Pi=\varphi \Psi\)
and ?ops \(=\left[\varphi_{O}{ }^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow o p s^{\prime}\right]\)
and \(? s^{\prime}=\varphi_{S} \Psi s\)
- TODO refactor.
\{
fix \(o p^{\prime}\)
assume \(o p^{\prime} \in\) set ops \({ }^{\prime}\)
moreover obtain \(o p\) where \(o p \in\) set ? ops and \(o p=\varphi_{O}{ }^{-1} \Psi o p^{\prime}\)
using calculation
by force
moreover obtain \(o p^{\prime \prime}\) where \(o p^{\prime \prime} \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)and \(o p^{\prime}=\varphi_{O} \Psi o p^{\prime \prime}\)
using assms(4) calculation(1)
by auto
moreover have is-valid-operator-sas-plus \(\Psi\) op"
using is-valid-problem-sas-plus-then(2) assms(1) calculation(4) unfolding is-valid-operator-sas-plus-def
by auto
moreover have \(o p=o p^{\prime \prime}\)
using sas-plus-operator-inverse-is[OF assms(1)] calculation(3, 4, 5)
by blast
ultimately have \(\exists o p \in\) set ?ops. op \(\in\) set ?ops \(\wedge o p=\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\)
\(\wedge\) is-valid-operator-sas-plus \(\Psi\) op
by blast
\(\}\) note \(n b_{1}=t h i s\)
have \(n b_{2}: \forall(v, a) \in \operatorname{dom}\) ? \(s^{\prime}\).
\(\forall\left(v, a^{\prime}\right) \in \operatorname{dom} ? s^{\prime}\).
\(? s^{\prime}(v, a)=\) Some True \(\wedge ? s^{\prime}\left(v, a^{\prime}\right)=\) Some True
\(\longrightarrow(v, a)=\left(v, a^{\prime}\right)\)
using state-to-strips-state-effect-consistent[OF assms(1), of --s]
by blast
\{
fix \(o p\)
assume \(o p \in\) set ?ops
hence map-of (precondition-of op) \(\subseteq_{m} s\)
using assms(5)
unfolding are-all-operators-applicable-in-def
is-operator-applicable-in-def list-all-iff
by blast
\(\}\) note \(n b_{3}=\) this
\{
fix \(o p^{\prime}\)
assume \(o p^{\prime} \in\) set ops \({ }^{\prime}\)
then obtain op where op-in-ops: op \(\in\) set ?ops
and op-is: op \(=\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\)
and is-valid-operator-op: is-valid-operator-sas-plus \(\Psi\) op
using \(n b_{1}\)
by force
moreover have preconditions-are-consistent:
\(\forall(v, a) \in \operatorname{set}\left(\right.\) precondition-of op). \(\forall\left(v^{\prime}, a^{\prime}\right) \in \operatorname{set}(\) precondition-of op). \(v \neq\)
\(v^{\prime} \vee a=a^{\prime}\)
using is-valid-operator-sas-plus-then(5) calculation(3)
unfolding is-valid-operator-sas-plus-def
by fast
moreover \{
fix \(v a\)
assume \((v, a) \in\) set (strips-operator.precondition-of op \({ }^{\prime}\) )
moreover have \(v\)-a-in-precondition-of-op: \((v, a) \in\) set (precondition-of op)
using op-is calculation
unfolding SAS-Plus-STRIPS.strips-op-to-sasp-def
strips-op-to-sasp-def
by auto
moreover have map-of (precondition-of op) \(v=\) Some \(a\) using map-of-constant-assignments-defined-if[OF
preconditions-are-consistent calculation(2)] by blast
moreover have \(s-o f-v-i s: s\) Some \(a\)
using \(n b_{3}[\) OF op-in-ops] calculation(3)
unfolding map-le-def
by force
moreover have \(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)and \(a \in \mathcal{R}_{+} \Psi v\)
using is-valid-operator-sas-plus-then (1, 2) is-valid-operator-op \(v\)-a-in-precondition-of-op
unfolding is-valid-operator-sas-plus-def
SAS-Plus-Representation.is-valid-operator-sas-plus-def Let-def list-all-iff
ListMem-iff
by auto+
moreover have \((v, a) \in d o m ? s^{\prime}\)
using state-to-strips-state-dom-is[OF assms(1)] s-of-v-is
calculation
by \(\operatorname{simp}\)
moreover have \(\left(\varphi_{S}{ }^{-1} \Psi ? s^{\prime}\right) v=\) Some a
using strips-state-to-state-inverse-is \([O F \operatorname{assms}(1,2,3)] s\)-of- \(v\)-is by argo
- TODO slow.
ultimately have ? \(s^{\prime}(v, a)=\) Some True
using strips-state-to-state-range-is[OF assms(1)] nb \(b_{2}\)
by auto
\}
ultimately have \(\forall(v, a) \in\) set (strips-operator.precondition-of op'). ?s \({ }^{\prime}(v, a)\)
= Some True
by fast
\}
thus ?thesis
unfolding are-all-operators-applicable-def is-operator-applicable-in-def
STRIPS-Representation.is-operator-applicable-in-def list-all-iff
by \(\operatorname{simp}\)
qed
lemma sas-plus-equivalent-to-strips-i-a-VII:
assumes is-valid-problem-sas-plus \(\Psi\)
and \(\operatorname{dom} s \subseteq \operatorname{set}\left((\Psi) \mathcal{V}_{+}\right)\)
and \(\forall v \in\) dom s. the \((s v) \in \mathcal{R}_{+} \Psi v\)
and set ops \({ }^{\prime} \subseteq \operatorname{set}\left((\varphi \Psi)_{\mathcal{O}}\right)\)
and are-all-operators-applicable-in s \(\left[\varphi_{O}{ }^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow o p s\right] \wedge\)
are-all-operator-effects-consistent \(\left[\varphi_{O}{ }^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow o p s\right]\)
shows STRIPS-Semantics.are-all-operator-effects-consistent ops'
proof -
let ? \(s^{\prime}=\varphi_{S} \Psi s\)
and ?ops \(=\left[\varphi_{O}{ }^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow o p s^{\prime}\right]\)
and \(? D=\) range-of \(\Psi\)
and \(? \Pi=\varphi \Psi\)
```

- TODO refactor.
{
fix }o\mp@subsup{p}{}{\prime
assume op'\in set ops'
moreover obtain op where op set ?ops and op = \varphi (O
using calculation
by force
moreover obtain }o\mp@subsup{p}{}{\prime\prime}\mathrm{ where }o\mp@subsup{p}{}{\prime\prime}\in\operatorname{set}((\Psi\mp@subsup{)}{\mp@subsup{\mathcal{O+}}{+}{\prime}}{})\mathrm{ and }o\mp@subsup{p}{}{\prime}=\mp@subsup{\varphi}{O}{}\Psio\mp@subsup{p}{}{\prime\prime
using assms(4) calculation(1)
by auto
moreover have is-valid-operator-sas-plus \Psi op"
using is-valid-problem-sas-plus-then(2) assms(1) calculation(4)
unfolding is-valid-operator-sas-plus-def
by auto
moreover have op =op"
using sas-plus-operator-inverse-is[OF assms(1)] calculation(3, 4, 5)
by blast
ultimately have }\existsop\in\mathrm{ set ?ops. op }\in\mathrm{ set ?ops }\wedge o\mp@subsup{p}{}{\prime}=(\mp@subsup{\varphi}{O}{}\Psiop
\wedge ~ i s - v a l i d - o p e r a t o r - s a s - p l u s ~ \Psi ~ o p ~
by blast
} note nb
{
fix op, ' }o\mp@subsup{p}{2}{\prime
assume o\mp@subsup{p}{1}{\prime}}\mp@subsup{}{}{\prime}\in\mathrm{ set ops'
and op }\mp@subsup{}{2}{\prime}\mp@subsup{}{}{\prime}\in\mathrm{ set ops'
and \exists(v,a)\inset (add-effects-of o\mp@subsup{p}{1}{\prime}). \exists (v', a') \in set (delete-effects-of o\mp@subsup{p}{2}{\prime}}\mp@subsup{}{}{\prime})
(v,a) = (v', a')
moreover obtain op op op
where op
and o\mp@subsup{p}{1}{\prime}}=\mp@subsup{\varphi}{O}{}\Psi\quad\Psi\mp@subsup{p}{1}{
and is-valid-operator-sas-plus \Psi op
and op, }\in\mathrm{ set ?ops
and o\mp@subsup{p}{2}{\prime}}\mp@subsup{}{}{\prime}=\mp@subsup{\varphi}{O}{O}\Psio\mp@subsup{p}{2}{
and is-valid-op p
using nb calculation(1, 2)
by meson
moreover obtain v v' a a
where (v,a) \in set (add-effects-of op }\mp@subsup{}{1}{\prime}
and (v', a') \in set (delete-effects-of op 2}\mp@subsup{}{}{\prime}
and (v,a)=( v',a}
using calculation
by blast
moreover have (v,a)\in set (effect-of op ( )
using calculation(5, 10)
unfolding SAS-Plus-STRIPS.sasp-op-to-strips-def
sasp-op-to-strips-def
by fastforce
moreover have v=\mp@subsup{v}{}{\prime}}\mathrm{ and }a=\mp@subsup{a}{}{\prime
using calculation(12)

```
by \(\operatorname{simp}+\)
- The next proof block shows that \(\left(v^{\prime}, a^{\prime}\right)\) is constructed from an effect \(\left(v^{\prime \prime}\right.\), \(a^{\prime \prime}\) ) s.t. \(a^{\prime} \neq a^{\prime \prime}\).
moreover \{
```

    have \(\left(v^{\prime}, a^{\prime}\right) \in\left(\bigcup\left(v^{\prime \prime}, a^{\prime \prime}\right) \in \operatorname{set}(\right.\) effect-of op 2\()\).
        \(\left.\left\{\left(v^{\prime \prime}, a^{\prime \prime \prime}\right) \mid a^{\prime \prime \prime} \cdot a^{\prime \prime \prime} \in\left(\mathcal{R}_{+} \Psi v^{\prime \prime}\right) \wedge a^{\prime \prime \prime} \neq a^{\prime \prime}\right\}\right)\)
        using sasp-op-to-strips-set-delete-effects-is
        calculation \((8,11)\) is-valid-op 2
        by blast
    then obtain \(v^{\prime \prime} a^{\prime \prime}\) where \(\left(v^{\prime \prime}, a^{\prime \prime}\right) \in \operatorname{set}\left(\right.\) effect-of op \(\left.p_{2}\right)\)
        and \(\left(v^{\prime}, a^{\prime}\right) \in\left\{\left(v^{\prime \prime}, a^{\prime \prime \prime}\right) \mid a^{\prime \prime \prime} . a^{\prime \prime \prime} \in\left(\mathcal{R}_{+} \Psi v^{\prime \prime}\right) \wedge a^{\prime \prime \prime} \neq a^{\prime \prime}\right\}\)
        by blast
    moreover have \(\left(v^{\prime}, a^{\prime \prime}\right) \in \operatorname{set}\left(\right.\) effect-of \(\left.o p_{2}\right)\)
        using calculation
        by blast
    moreover have \(a^{\prime} \in \mathcal{R}_{+} \Psi v^{\prime \prime}\) and \(a^{\prime} \neq a^{\prime \prime}\)
        using calculation(1, 2)
        by fast+
    ultimately have \(\exists a^{\prime \prime} .\left(v^{\prime}, a^{\prime \prime}\right) \in \operatorname{set}\left(\right.\) effect-of op \(\left.p_{2}\right) \wedge a^{\prime} \in\left(\mathcal{R}_{+} \Psi v^{\prime}\right)\)
    \(\wedge a^{\prime} \neq a^{\prime \prime}\)
        by blast
    \}
moreover obtain $a^{\prime \prime}$ where $\left(v^{\prime}, a^{\prime \prime}\right) \in$ set (effect-of op ${ }_{2}$ )
and $a^{\prime} \in \mathcal{R}_{+} \Psi v^{\prime}$
and $a^{\prime} \neq a^{\prime \prime}$
using calculation(16)
by blast
moreover have $\exists(v, a) \in \operatorname{set}\left(\right.$ effect-of op $\left.p_{1}\right) .\left(\exists\left(v^{\prime}, a^{\prime}\right) \in \operatorname{set}\left(\right.\right.$ effect-of op $\left.p_{2}\right)$.
$\left.v=v^{\prime} \wedge a \neq a^{\prime}\right)$
using calculation(13, 14, 15, 17, 19)
by blast
moreover have $\neg$ are-operator-effects-consistent $o p_{1} o p_{2}$
unfolding are-operator-effects-consistent-def list-all-iff
using calculation(20)
by fastforce
ultimately have $\neg$ are-all-operator-effects-consistent ?ops
unfolding are-all-operator-effects-consistent-def list-all-iff
by meson
$\}$ note $n b_{2}=$ this
fix $o p_{1}{ }^{\prime} o p_{2}{ }^{\prime}$
assume $o p_{1}{ }^{\prime}$-in-ops: $o p_{1}{ }^{\prime} \in$ set ops ${ }^{\prime}$ and $o p_{2}{ }^{\prime}$-in-ops: $o p_{2}{ }^{\prime} \in$ set ops ${ }^{\prime}$
have STRIPS-Semantics.are-operator-effects-consistent op ${ }_{1}{ }^{\prime} o p_{2}{ }^{\prime}$
proof (rule ccontr)
assume $\neg$ STRIPS-Semantics.are-operator-effects-consistent op ${ }_{1}{ }^{\prime} o p_{2}{ }^{\prime}$
then consider $(A) \exists(v, a) \in \operatorname{set}\left(a d d\right.$-effects-of op $\left.{ }_{1}{ }^{\prime}\right)$.
$\exists\left(v^{\prime}, a^{\prime}\right) \in \operatorname{set}\left(\right.$ delete-effects-of op $\left.{ }_{2}{ }^{\prime}\right) .(v, a)=\left(v^{\prime}, a^{\prime}\right)$
$\mid(B) \exists(v, a) \in \operatorname{set}\left(a d d-e f f e c t s\right.$-of $\left.o p_{2}{ }^{\prime}\right)$.

```
\{
```

        \exists(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\operatorname{set}(\mathrm{ delete-effects-of op }\mp@subsup{}{1}{\prime}).(v,a)=(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})
        unfolding STRIPS-Semantics.are-operator-effects-consistent-def list-ex-iff
        by fastforce
        thus False
            using nb [ [OF op 1'-in-ops op 2'-in-ops] nb [ [OF op ''-in-ops op }\mp@subsup{}{1}{\prime}\mp@subsup{}{}{\prime}\mathrm{ -in-ops]
    assms(5)
by (cases, argo, force)
qed
}
thus ?thesis
unfolding STRIPS-Semantics.are-all-operator-effects-consistent-def
STRIPS-Semantics.are-operator-effects-consistent-def list-all-iff
by blast
qed
lemma sas-plus-equivalent-to-strips-i-a-VIII:
assumes is-valid-problem-sas-plus \Psi
and dom s\subseteqset ((\Psi)\mp@subsup{\mathcal{V}}{+}{*})
and}\forallv\indom s. the (sv)\in\mathcal{R}+\Psi
and set ops'}\subseteq\operatorname{set}((\varphi\Psi\mp@subsup{)}{\mathcal{O}}{\prime}
and are-all-operators-applicable-in s [ [ }\mp@subsup{\rho}{O}{-1}\Psi \Psip'.op' '\leftarrowops']
are-all-operator-effects-consistent [ }\mp@subsup{\varphi}{O}{-1}\Psi \Psio\mp@subsup{p}{}{\prime}.o\mp@subsup{p}{}{\prime}\leftarrow\leftarrowops'
shows STRIPS-Semantics.are-all-operators-applicable ( }\mp@subsup{\varphi}{S}{}\Psi s) ops
^ STRIPS-Semantics.are-all-operator-effects-consistent ops'
using sas-plus-equivalent-to-strips-i-a-VI sas-plus-equivalent-to-strips-i-a-VII assms
by fastforce
lemma sas-plus-equivalent-to-strips-i-a-IX:
assumes dom s\subseteqV
and }\forallop\in\mathrm{ set ops.}\forall(v,a)\in\mathrm{ set (effect-of op). v}\in
shows dom (execute-parallel-operator-sas-plus s ops) \subseteqV
proof -
show ?thesis
using assms
proof (induction ops arbitrary: s)
case Nil
then show ?case
unfolding execute-parallel-operator-sas-plus-def
by simp
next
case (Cons op ops)
let ? }\mp@subsup{s}{}{\prime}=s++\mathrm{ map-of (effect-of op)
- TODO Wrap IH instantiation in block.
{
have }\forall(v,a)\in\operatorname{set}(effect-of op).v\in
using Cons.prems(2)
by fastforce
moreover have fst'set (effect-of op) \subseteqV

```
```

            using calculation
            by fastforce
            ultimately have dom?s'\subseteqV
            unfolding dom-map-add dom-map-of-conv-image-fst
            using Cons.prems(1)
            by blast
    }
    moreover have }\forallop\in\mathrm{ set ops. }\forall(v,a)\in\mathrm{ set (effect-of op). v}\in
            using Cons.prems(2)
            by fastforce
    ultimately have dom (execute-parallel-operator-sas-plus ?s' ops)\subseteqV
        using Cons.IH[of ?s']
        by fast
    thus ?case
        unfolding execute-parallel-operator-sas-plus-cons.
    qed
    qed

```
- NOTE Show that the domain value constraint on states is monotonous w.r.t. to valid operator execution. I.e. if a parallel operator is executed on a state for which the domain value constraint holds, the domain value constraint will also hold on the resultant state.
```

lemma sas-plus-equivalent-to-strips-i-a-X:
assumes dom $s \subseteq V$
and $V \subseteq \operatorname{dom} D$
and $\forall v \in$ dom s. the $(s v) \in$ set (the $(D v))$
and $\forall o p \in$ set ops. $\forall(v, a) \in \operatorname{set}(e f f e c t-o f$ op). $v \in V \wedge a \in \operatorname{set}($ the $(D v))$
shows $\forall v \in$ dom (execute-parallel-operator-sas-plus s ops).
the (execute-parallel-operator-sas-plus s ops $v$ ) $\in \operatorname{set}($ the $(D v))$
proof -
show ?thesis
using assms
proof (induction ops arbitrary: s)
case Nil
then show ?case
unfolding execute-parallel-operator-sas-plus-def
by $\operatorname{simp}$
next
case (Cons op ops)
let ? $s^{\prime}=s++$ map-of (effect-of op)
\{
\{
have $\forall(v, a) \in \operatorname{set}($ effect-of op). $v \in V$
using Cons.prems(4)
by fastforce
moreover have $f s t$ 'set (effect-of op) $\subseteq V$
using calculation
by fastforce

```
```

    ultimately have dom? 's}\subseteq\subseteq
            unfolding dom-map-add dom-map-of-conv-image-fst
            using Cons.prems(1)
            by blast
        }
        moreover {
            fix v
            assume v-in-dom-s':}v\indom?s
            hence the (?s'v)\in set (the (Dv))
            proof (cases v dom (map-of (effect-of op)))
            case True
            moreover have ? s' v=(map-of (effect-of op)) v
                unfolding map-add-dom-app-simps(1)[OF True]
                by blast
            moreover obtain a where (map-of (effect-of op)) v=Some a
                using calculation(1)
                by fast
            moreover have (v,a)\in set (effect-of op)
                using map-of-SomeD calculation(3)
                by fast
            moreover have a\inset (the (Dv))
                using Cons.prems(4) calculation(4)
                by fastforce
            ultimately show ?thesis
                by force
            next
            case False
            then show ?thesis
                    unfolding map-add-dom-app-simps(3)[OF False]
                    using Cons.prems(3) v-in-dom-s'
                by fast
            qed
        }
        moreover have }\forallop\in set ops. \forall(v,a)\in set (effect-of op). v\inV\wedgea
    set (the (D v))
using Cons.prems(4)
by auto
ultimately have }\forallv\indom (execute-parallel-operator-sas-plus ?s' ops)
the (execute-parallel-operator-sas-plus ?s' ops v)\in set (the (D v))
using Cons.IH[of s ++ map-of (effect-of op),OF - Cons.prems(2)]
by meson
}
thus ?case
unfolding execute-parallel-operator-sas-plus-cons
by blast
qed
qed
lemma transfom-sas-plus-problem-to-strips-problem-operators-valid:

```
```

    assumes is-valid-problem-sas-plus \Psi
    and o\mp@subsup{p}{}{\prime}\in\operatorname{set}((\varphi\Psi\mp@subsup{)}{\mathcal{O}}{})
    obtains op
    where op \in set ((\Psi) ()
    and op'}=(\mp@subsup{\varphi}{O}{}\Psiop) is-valid-operator-sas-plus \Psi o
    proof -
{
obtain op where op set ((\Psi) (\Psi\mp@subsup{O}{+}{}) and op' = \varphi
using assms
by auto
moreover have is-valid-operator-sas-plus \Psi op
using is-valid-problem-sas-plus-then(2) assms(1) calculation(1)
by auto
ultimately have }\existsop\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O+}}{}).o\mp@subsup{p}{}{\prime}=(\mp@subsup{\varphi}{O}{}\Psiop
\wedge ~ i s - v a l i d - o p e r a t o r - s a s - p l u s ~ \Psi ~ o p ~
by blast
}
thus ?thesis
using that
by blast
qed
lemma sas-plus-equivalent-to-strips-i-a-XI:
assumes is-valid-problem-sas-plus \Psi
and o\mp@subsup{p}{}{\prime}\in\operatorname{set}((\varphi\Psi\mp@subsup{)}{\mathcal{O}}{})
shows ( }\mp@subsup{\varphi}{S}{}\Psi\mp@code{s)++ map-of (effect-to-assignments op')
= \varphi S \Psi (s++ map-of (effect-of ( }\mp@subsup{\varphi}{O}{O}\mp@subsup{}{}{-1}\Psio\mp@subsup{p}{}{\prime}))
proof -
let ?\Pi=\varphi\Psi
let ?vs = variables-of }
and?ops = operators-of \Psi
and ?ops' = strips-problem.operators-of ?\Pi
let ? s' = \varphi S \Psi s
let ?t = ?s' ++ map-of (effect-to-assignments op')
and ?t'}=\mp@subsup{\varphi}{S}{}\Psi(s++\mathrm{ map-of (effect-of ( }\mp@subsup{\varphi}{O}{-1}\Psi \Psio\mp@subsup{p}{}{\prime}))
obtain op where op'-is:op' }=(\mp@subsup{\varphi}{O}{}
and op-in-ops:op }\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O}+}{}
and is-valid-operator-op: is-valid-operator-sas-plus \Psi op
using transfom-sas-plus-problem-to-strips-problem-operators-valid[OF assms]
by auto
have n\mp@subsup{b}{1}{}:(\mp@subsup{\varphi}{O}{-1}\Psio\mp@subsup{p}{}{\prime})=op
using sas-plus-operator-inverse-is[OF assms(1)] op'-is op-in-ops
by blast
- TODO refactor.
{
have dom (map-of (effect-to-assignments op'))
=set (strips-operator.add-effects-of op') \cup set (strips-operator.delete-effects-of
op')

```

\section*{unfolding dom-map-of-conv-image-fst}
by force
- TODO slow.
also have \(\ldots=\) set \(\left(\right.\) effect-of op) \(\cup\) set (strips-operator.delete-effects-of op \({ }^{\prime}\) )
using \(o p^{\prime}-i s\)
unfolding SAS-Plus-STRIPS.sasp-op-to-strips-def
sasp-op-to-strips-def
by auto
- TODO slow.
finally have dom (map-of (effect-to-assignments op')) \(=\) set (effect-of op)
\(\cup\left(\bigcup(v, a) \in \operatorname{set}(\right.\) effect-of op \(\left.) .\left\{\left(v, a^{\prime}\right) \mid a^{\prime} \cdot a^{\prime} \in\left(\mathcal{R}_{+} \Psi v\right) \wedge a^{\prime} \neq a\right\}\right)\)
using sasp-op-to-strips-set-delete-effects-is[OF
is-valid-operator-op] \(o p^{\prime}-i s\)
by argo
\(\}\) note \(n b_{2}=\) this
have \(n b_{3}\) : dom ?t \(=\) dom ? \(s^{\prime} \cup\) set \((\) effect-of op)
\(\cup\left(\bigcup(v, a) \in \operatorname{set}(\right.\) effect-of op \(\left.) .\left\{\left(v, a^{\prime}\right) \mid a^{\prime} . a^{\prime} \in\left(\mathcal{R}_{+} \Psi v\right) \wedge a^{\prime} \neq a\right\}\right)\)
unfolding \(n b_{2}\) dom-map-add
by blast
- TODO refactor.
have \(n b_{4}\) : dom ( \(s++\) map-of (effect-of \(\left(\varphi_{O}{ }^{-1} \Psi\right.\) op \(\left.\left.{ }^{\prime}\right)\right)\) )
\(=\operatorname{dom} s \cup f s t\) 'set (effect-of op)
unfolding dom-map-add dom-map-of-conv-image-fst \(n b_{1}\)
by fast
\{
let ? \(u=s++\) map-of (effect-of \(\left.\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\right)\)
have dom ? \(t^{\prime}=\left(\bigcup v \in\left\{v \mid v . v \in \operatorname{set}\left((\Psi)_{\mathcal{V}+}\right) \wedge ? u v \neq\right.\right.\) None \(\}\).
\(\left.\left\{(v, a) \mid a . a \in \mathcal{R}_{+} \Psi v\right\}\right)\)
using state-to-strips-state-dom-is[OF assms(1)]
by presburger
\} note \(n b_{5}=\) this
- TODO refactor.
have \(n b_{6}\) : set (add-effects-of op \(\left.{ }^{\prime}\right)=\) set (effect-of op)
using \(o p^{\prime}\)-is
unfolding SAS-Plus-STRIPS.sasp-op-to-strips-def sasp-op-to-strips-def
by auto
- TODO refactor.
have \(n b_{7}\) : set (delete-effects-of op \(\left.{ }^{\prime}\right)=(\bigcup(v, a) \in\) set (effect-of op).
\(\left.\left\{\left(v, a^{\prime}\right) \mid a^{\prime} . a^{\prime} \in\left(\mathcal{R}_{+} \Psi v\right) \wedge a^{\prime} \neq a\right\}\right)\)
using sasp-op-to-strips-set-delete-effects-is[OF is-valid-operator-op] \(o p^{\prime}\)-is
by argo
- TODO refactor.
\{
let ?Add \(=\) set (effect-of op)
let ?Delete \(=(\bigcup(v, a) \in\) set (effect-of op).
\(\left.\left\{\left(v, a^{\prime}\right) \mid a^{\prime} . a^{\prime} \in\left(\mathcal{R}_{+} \Psi v\right) \wedge a^{\prime} \neq a\right\}\right)\)
have dom-add: dom (map-of (map ( \(\lambda v\). \((v\), True \())\left(\right.\) add-effects-of op \(\left.\left.\left.{ }^{\prime}\right)\right)\right)=\) ?Add
```

        unfolding dom-map-of-conv-image-fst set-map image-comp comp-apply
        using nb
        by simp
    have dom-delete: dom (map-of (map (\lambdav. (v, False)) (delete-effects-of op}\mp@subsup{}{\prime}{\prime}))
    ?Delete
unfolding dom-map-of-conv-image-fst set-map image-comp comp-apply
using nb
by auto
{
{
fix va
assume v-a-in-dom-add: (v,a)\indom(map-of (map (\lambdav. (v,True))
(add-effects-of op')))
have (v,a)\not\indom(map-of (map (\lambdav. (v, False)) (delete-effects-of op')))
proof (rule ccontr)
assume }\neg((v,a)\not\in\mathrm{ dom (map-of (map ( \v. (v, False)) (delete-effects-of
op}\mp@subsup{}{}{\prime})))
then have (v,a)\in?Delete and (v,a)\in?Add
using dom-add dom-delete v-a-in-dom-add
by argo+
moreover have }\forall(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in?Add.v\not=\mp@subsup{v}{}{\prime}\veea=\mp@subsup{a}{}{\prime
using is-valid-operator-sas-plus-then(6) is-valid-operator-op
calculation(2)
unfolding is-valid-operator-sas-plus-def
by fast
ultimately show False
by fast
qed
}
hence disjnt (dom (map-of (map ( }\lambdav.(v,True)) (add-effects-of op'))))
(dom (map-of (map (\lambdav. (v, False)) (delete-effects-of op}\mp@subsup{}{}{\prime})))
unfolding disjnt-def Int-def
using }n\mp@subsup{b}{7}{
by simp
}
hence dom (map-of (map (\lambdav. (v, True)) (add-effects-of op'))) = ?Add
and dom (map-of (map (\lambdav. (v, False)) (delete-effects-of op'))) = ?Delete
and disjnt (dom (map-of (map (\lambdav. (v, True)) (add-effects-of op'))))
(dom (map-of (map (\lambdav. (v, False)) (delete-effects-of op }\mp@subsup{}{}{\prime}))
using dom-add dom-delete
by blast+
} note nb }=thi

- TODO refactor.
{
let ?Add = set (effect-of op)
let ?Delete =(U(v,a) \in set (effect-of op).
{(v,\mp@subsup{a}{}{\prime})|\mp@subsup{a}{}{\prime}.\mp@subsup{a}{}{\prime}\in(\mp@subsup{\mathcal{R}}{+}{}\Psiv)\wedge\mp@subsup{a}{}{\prime}\not=a})
    - TODO slow.

```
```

    have }\forall(v,a)\in\mathrm{ ?Add. map-of (effect-to-assignments op') (v,a)=Some True
    and }\forall(v,a)\in\mathrm{ ?Delete. map-of (effect-to-assignments op') (v,a)=Some
    False
proof -
{
fix va
assume (v,a)\in?Add
hence map-of (effect-to-assignments op') (v,a)=Some True
unfolding effect-to-assignments-simp
using n\mp@subsup{b}{6}{}\mathrm{ map-of-defined-if-constructed-from-list-of-constant-assignments[of}
map (\lambdav. (v, True)) (add-effects-of op') True add-effects-of op']
by force
}
moreover {
fix va
assume (v,a) \in ?Delete
moreover have (v,a)\indom (map-of (map (\lambdav. (v, False)) (delete-effects-of
op')))
using nb (2) calculation(1)
by argo
moreover have (v,a)\not\indom (map-of (map (\lambdav. (v, True)) (add-effects-of
op')))
using nb
unfolding disjnt-def
using calculation(1)
by blast
moreover have map-of (effect-to-assignments op') (v,a)
=map-of (map (\lambdav. (v, False)) (delete-effects-of op')) (v,a)
unfolding effect-to-assignments-simp map-of-append
using map-add-dom-app-simps(3)[OF calculation(3)]
by presburger
TODO slow.
ultimately have map-of (effect-to-assignments op') (v,a)=Some False
using map-of-defined-if-constructed-from-list-of-constant-assignments[
of map ( }\lambdav.(v,\mathrm{ False)) (delete-effects-of op') False delete-effects-of op']
nb
by auto
}
ultimately show }\forall(v,a)\in\mathrm{ ?Add. map-of (effect-to-assignments op') (v,
a)= Some True
and }\forall(v,a)\in\mathrm{ ?Delete. map-of (effect-to-assignments op') (v,a)=Some
False
by blast+
qed
} note nb9 = this
{
fix va
assume (v,a)\in set (effect-of op)

```
moreover have \(\forall(v, a) \in \operatorname{set}\left(\right.\) effect-of op). \(\forall\left(v^{\prime}, a^{\prime}\right) \in\) set (effect-of op). \(v \neq\) \(v^{\prime} \vee a=a^{\prime}\)
using is-valid-operator-sas-plus-then is-valid-operator-op
unfolding is-valid-operator-sas-plus-def
by fast
ultimately have map-of (effect-of op) \(v=\) Some a
using map-of-constant-assignments-defined-if[of effect-of op]
by presburger
\} note \(n b_{10}=\) this
\{
fix \(v a\)
assume \(v\)-a-in-effect-of-op: \((v, a) \in\) set (effect-of op)
and \(\left(s++\right.\) map-of (effect-of \(\left.\left.\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\right)\right) v \neq\) None
moreover have \(v \in\) set ?vs
using is-valid-operator-op is-valid-operator-sas-plus-then(3) calculation(1)
by fastforce
moreover \{
have is-valid-problem-strips ?
using is-valid-problem-sas-plus-then-strips-transformation-too
\(\operatorname{assms}\) (1)
by blast
thm calculation(1) \(n b_{6}\) assms(2)
moreover have set (add-effects-of op') \(\subseteq\) set \(\left((? \Pi)_{\mathcal{V}}\right)\)
using assms(2) is-valid-problem-strips-operator-variable-sets(2)
calculation
by blast
moreover have \((v, a) \in \operatorname{set}((? \Pi) \mathcal{V})\)
using \(v\)-a-in-effect-of-op \(n b_{6}\) calculation(2)
by blast
ultimately have \(a \in \mathcal{R}_{+} \Psi v\)
using sas-plus-problem-to-strips-problem-variable-set-element-iff[OF \(\operatorname{assms}(1)]\)
by fast
\}
- TODO slow.
ultimately have \((v, a) \in \operatorname{dom}\left(\varphi_{S} \Psi\left(s++\operatorname{map-of}\left(\operatorname{effect-of}\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\right)\right)\right)\)
using state-to-strips-state-dom-is[OF assms(1), of
\[
\left.s++\operatorname{map}-o f\left(\text { effect-of }\left(\varphi_{O}^{-1} \Psi o p^{\prime}\right)\right)\right]
\]
by simp
\(\}\) note \(n b_{11}=t h i s\)
\{
fix \(v a\)
assume \((v, a) \in\) set (effect-of op)
moreover have \(v \in\) dom (map-of (effect-of op))
unfolding dom-map-of-conv-image-fst
using calculation
by force
moreover have \(\left(s++\right.\) map-of (effect-of \(\left(\varphi_{O^{-1}} \Psi\right.\) op \(\left.\left.)\right)\right) v=\) Some a
unfolding map-add-dom-app-simps(1)[OF calculation(2)] \(n b_{1}\)
using \(n b_{10}\) calculation(1)
by blast
moreover have \(\left(s++\right.\) map-of (effect-of \(\left.\left.\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\right)\right) v \neq\) None
using calculation(3)
by auto
moreover have \((v, a) \in \operatorname{dom}\left(\varphi_{S} \Psi\left(s++\right.\right.\) map-of \(\left(\right.\) effect-of \(\left.\left.\left.\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\right)\right)\right)\)
using \(n b_{11}\) calculation \((1,4)\)
by presburger
ultimately have \(\left(\varphi_{S} \Psi\left(s++\operatorname{map}-\right.\right.\) of \(\left.\left.\left(\operatorname{effect-of}\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\right)\right)\right)(v, a)=\)
Some True
using state-to-strips-state-range-is[OF assms(1)]
by \(\operatorname{simp}\)
\(\}\) note \(n b_{12}=\) this
\{
fix \(v a^{\prime}\)
assume \(\left(v, a^{\prime}\right) \in \operatorname{dom}\) (map-of (effect-to-assignments op \(\left.{ }^{\prime}\right)\) )
and \(\left(v, a^{\prime}\right) \in(\bigcup(v, a) \in \operatorname{set}(\) effect-of op). \(\left.\left\{\left(v, a^{\prime}\right) \mid a^{\prime} \cdot a^{\prime} \in\left(\mathcal{R}_{+} \Psi v\right) \wedge a^{\prime} \neq a\right\}\right)\)
moreover have \(v \in\) dom (map-of (effect-of op))
unfolding dom-map-of-conv-image-fst
using calculation(2)
by force
moreover have \(v \in\) set ?vs
using calculation(3) is-valid-operator-sas-plus-then(3) is-valid-operator-op
unfolding dom-map-of-conv-image-fst is-valid-operator-sas-plus-def
by fastforce
moreover obtain \(a\) where \((v, a) \in \operatorname{set}\) (effect-of op)
and \(a^{\prime} \in \mathcal{R}_{+} \Psi v\)
and \(a^{\prime} \neq a\)
using calculation(2)
by blast
moreover have \(\left(s++\right.\) map-of (effect-of \(\left.\left.\left(\varphi_{O^{-1}} \Psi o p^{\prime}\right)\right)\right) v=\) Some a
unfolding map-add-dom-app-simps(1)[OF calculation(3)] \(n b_{1}\)
using \(n b_{10}\) calculation(5)
by blast
moreover have \(\left(s++\right.\) map-of (effect-of \(\left(\varphi_{O^{-1}} \Psi\right.\) op \(\left.\left.)\right)\right) v \neq\) None
using calculation(8)
by auto
- TODO slow.
moreover have \(\left(v, a^{\prime}\right) \in \operatorname{dom}\left(\varphi_{S} \Psi\left(s++\operatorname{map-of}\left(\operatorname{effect-of}\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\right)\right)\right)\)
using state-to-strips-state-dom-is[OF assms(1), of \(s++\) map-of (effect-of \(\left(\varphi_{O}{ }^{-1} \Psi\right.\) op \(\left.{ }^{\prime}\right)\) )] calculation \((4,6,9)\)
by \(\operatorname{simp}\)
- TODO slow.
ultimately have \(\left(\varphi_{S} \Psi\left(s++\right.\right.\) map-of \(\left.\left.\left(\operatorname{effect-of}\left(\varphi_{O}^{-1} \Psi o p^{\prime}\right)\right)\right)\right)\left(v, a^{\prime}\right)=\)
Some False
using state-to-strips-state-range-is[OF assms(1),
of \(v a^{\prime} s++\) map-of (effect-of \(\left.\left.\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\right)\right]\)
```

    by simp
    }ote nb 13}=thi
{
fix va
assume (v,a)\indom ?t
and (v,a)\not\indom(map-of (effect-to-assignments op'))
moreover have (v,a)\indom ?s
using calculation(1, 2)
unfolding dom-map-add
by blast
moreover have ?t (v,a)=?s'(v,a)
unfolding map-add-dom-app-simps(3)[OF calculation(2)]..
ultimately have ?t ( }v,a)=\mathrm{ Some (the (s v)=a)
using state-to-strips-state-range-is[OF assms(1)]
by presburger
} note nb 14 = this
{
fix va
assume (v,a)\indom ?t
and v-a-not-in: (v,a)\not\indom (map-of (effect-to-assignments op'))
moreover have (v,a)\indom? ? }\mp@subsup{}{}{\prime
using calculation(1, 2)
unfolding dom-map-add
by blast
moreover have (v,a)\in(\bigcupv\in{v|v.v\in\operatorname{set}((\Psi\mp@subsup{)}{\mp@subsup{\mathcal{V}}{+}{}}{})\wedgesv\not= None }.
{(v,a)|a.a\in\mathcal{R}
using state-to-strips-state-dom-is[OF assms(1)] calculation(3)
by presburger
moreover have v\in set ((\Psi)\mp@subsup{\mathcal{V}}{+}{})\mathrm{ and s v}\not=None and a\in\mathcal{R}
using calculation(4)
by blast+
- NOTE Hasn't this been proved before?
moreover {
have dom (map-of (effect-to-assignments op'))}=(\bigcup(v,a)\in\mathrm{ set (effect-of
op). {(v,a)})
\cup(\bigcup(v,a) \in set (effect-of op).
{(v,\mp@subsup{a}{}{\prime})|\mp@subsup{a}{}{\prime}.\mp@subsup{a}{}{\prime}\in(\mp@subsup{\mathcal{R}}{+}{}\Psiv)\wedge\mp@subsup{a}{}{\prime}\not=a})
unfolding n\mp@subsup{b}{2}{}
by blast
also have ... =(\bigcup(v,a)\inset (effect-of op). {(v,a)}
\cup \{ ( v , a ^ { \prime } ) \| a ^ { \prime } . a ^ { \prime } \in ( \mathcal { R } _ { + } \Psi v ) \wedge a ^ { \prime } \neq a \} )
by blast
finally have dom (map-of (effect-to-assignments op'))

```

```

            \cup \{ ( v , a ) \| a . a \in \mathcal { R } _ { + } \Psi v \} )
        by auto
    then have (v,a)\not\in(\bigcup(v,a)\in set (effect-of op).
        {(v,a)|a.a\in\mathcal{R}}+\Psiv}
        using v-a-not-in
    ```
```

    by blast
    }
- TODO slow.
moreover have v\not\in dom (map-of (effect-of op))
using dom-map-of-conv-image-fst calculation
by fastforce
moreover have (s++ map-of (effect-of ( (\varphiO}\mp@subsup{O}{}{-1}\Psi op'))) v=s
unfolding n\mp@subsup{b}{1}{} map-add-dom-app-simps(3)[OF calculation(9)]
by simp
- TODO slow.
moreover have (v,a)\indom ?t'
using state-to-strips-state-dom-is[OF assms(1), of
s++ map-of (effect-of ( }\mp@subsup{\varphi}{O}{-1}\Psi \Psiop'))] calculation(5, 6, 7, 8, 10)
by simp
ultimately have ? t' ( v,a) = Some (the (s v)=a)
using state-to-strips-state-range-is[OF assms(1)]
by presburger
} note nb 15 = this

- TODO refactor.
have n\mp@subsup{b}{16}{}}\mathrm{ : dom ?t = (\v { {v|v.v set (( (T)}\mp@subsup{\mathcal{V}}{+}{})\wedgesv\not= None }
{(v,a)|a.a\in(\mp@subsup{\mathcal{R}}{+}{}\Psiv)})
\cup \mp@code { s e t ~ ( e f f e c t - o f ~ o p ) }
\cup (U(v,a)\inset (effect-of op).
{(v,\mp@subsup{a}{}{\prime})|\mp@subsup{a}{}{\prime}.\mp@subsup{a}{}{\prime}\in(\mp@subsup{\mathcal{R}}{+}{}\Psiv)\wedge\mp@subsup{a}{}{\prime}\not=a})
unfolding dom-map-add n\mp@subsup{b}{2}{}
using state-to-strips-state-dom-is[OF assms(1), of s]
by auto
{
{
fix }v
assume (v,a)\indom ?t
then consider (A) (v,a) \in dom ( }\mp@subsup{\varphi}{S}{}\Psis
| (B) (v,a) \indom (map-of (effect-to-assignments op'))
by fast
hence (v,a)\indom? ? t'
proof (cases)
case }
then have v\in set ((\Psi)\mp@subsup{\mathcal{V}}{+}{})\mathrm{ and s v}=\mathrm{ None and }a\in\mp@subsup{\mathcal{R}}{+}{}\Psiv
unfolding state-to-strips-state-dom-element-iff[OF assms(1)]
by blast+
thm map-add-None state-to-strips-state-dom-element-iff[OF assms(1)]
moreover have (s++ map-of (effect-of ( }\mp@subsup{\varphi}{O}{-1}\Psi \Psiop'))) v\not= Non
using calculation(2)
by simp
ultimately show ?thesis
unfolding state-to-strips-state-dom-element-iff[OF assms(1)]
by blast
next
case B

```
```

        then have (v,a)\in
                set (effect-of op)
                \cup(\bigcup(v,a)\inset (effect-of op). {(v,\mp@subsup{a}{}{\prime})|\mp@subsup{a}{}{\prime}.\mp@subsup{a}{}{\prime}\in\mp@subsup{\mathcal{R}}{+}{}\Psiv\wedge\mp@subsup{a}{}{\prime}\not=a})
            unfolding nb 
            by blast
        then consider ( }\mp@subsup{B}{1}{})(v,a)\in\operatorname{set}(\mathrm{ effect-of op)
            | (B2) (v,a) \in (\bigcup(v,a) \inset (effect-of op).
            {(v,\mp@subsup{a}{}{\prime})|\mp@subsup{a}{}{\prime}.\mp@subsup{a}{}{\prime}\in\mathcal{R}}
            by blast
            thm nb l2 nb 13 nb 
            thus ?thesis
            proof (cases)
                case B1
                    then show ?thesis
                    using nb 
                    by fast
            next
                    case B2
                    then show ?thesis
                    using nb lu B
                by blast
            qed
        qed
    }
    moreover {
    let ?u =s++ map-of (effect-of ( }\mp@subsup{\varphi}{O}{-1}\Psi \Psiop')
    fix va
    assume v-a-in-dom-t':(v,a)\indom? ?'
    thm nb 
    then have v-in-vs: v\in\operatorname{set}((\Psi)\mp@subsup{\mathcal{V}}{+}{})
        and u-of-v-is-not-None: ?u v \not= None
        and a-in-range-of-v: a }\in\mp@subsup{\mathcal{R}}{+}{}\Psi
        using state-to-strips-state-dom-element-iff [OF assms(1)]
        v-a-in-dom-t'
    by meson+
    {
        assume (v,a)\not\indom ?t
        then have contradiction: (v,a) }\not
            (\bigcupv\in{v|v.v\inset ((\Psi)\mp@subsup{\mathcal{V}}{+}{})\wedgesv\not=None}.{(v,a)|a.a\in\mathcal{R}
    })
set (effect-of op)
\cup ( \bigcup ( v , a ) \in s e t ~ ( e f f e c t - o f ~ o p ) . \{ ( v , a ^ { \prime } ) \| a ^ { \prime } . a ^ { \prime } \in \mathcal { R } _ { + } \Psi v \wedge a ^ { \prime } \neq a \} )
unfolding nb
by fast
hence False
proof (cases map-of (effect-of ( ( }\mp@subsup{O}{O}{-1}\Psi %o ')) v = None)
case True
then have s v\not= None
using u-of-v-is-not-None

```
```

                    by simp
            then have }(v,a)\in(\bigcupv\in{v|v.v\in\operatorname{set}((\Psi\mp@subsup{)}{\mp@subsup{\mathcal{V}}{+}{\prime}}{})\wedgesv\not=None}
                    {(v,a)|a.a\in\mathcal{R}
                    using v-in-vs a-in-range-of-v
                    by blast
            thus ?thesis
                    using contradiction
                    by blast
        next
            case False
            then have v\indom (map-of (effect-of op))
            using u-of-v-is-not-None nb
            by blast
            then obtain }\mp@subsup{a}{}{\prime}\mathrm{ where map-of-effect-of-op-v-is:map-of (effect-of op)v
    =Some a'
by blast
then have v-a'-in: (v, a')\in set (effect-of op)
using map-of-SomeD
by fast
then show ?thesis
proof (cases a = a')
case True
then have (v,a)\in set (effect-of op)
using v-a'-in
by blast
then show ?thesis
using contradiction
by blast
next
case False
then have (v,a)\in(\bigcup(v,a)\inset (effect-of op).
{(v,a})|\mp@subsup{a}{}{\prime}.\mp@subsup{a}{}{\prime}\in\mp@subsup{\mathcal{R}}{+}{}\Psiv\wedge\mp@subsup{a}{}{\prime}\not=a}
using v-a'-in calculation a-in-range-of-v
by blast
thus ?thesis
using contradiction
by fast
qed
qed
}
hence (v,a)\indom ?t
by argo
}
moreover have dom ?t }\subseteqdom?t' and dom? ?t'\subseteqdom?
subgoal
using calculation(1) subrelI[of dom ?t dom ?t']
by fast
subgoal
using calculation(2) subrelI[of dom ?t' dom?t]

```
```

            by argo
        done
    ultimately have dom?t = dom ? 't'
    by force
    } note nb }\mp@subsup{b}{17}{}=\mathrm{ this
{
fix va
assume v-a-in-dom-t: (v,a)\indom?t
hence ?t (v,a)=? ?t' (v,a)
proof (cases (v,a)\indom (map-of (effect-to-assignments op')))
case True
- TODO slow.
- NOTE Split on the (disjunct) domain variable sets of map-of (effect-to-assignments
op')
then consider (A1) (v,a) \in set (effect-of op)
| (A2) (v,a) \in(\bigcup(v,a) \in set (effect-of op).
{(v,\mp@subsup{a}{}{\prime})|\mp@subsup{a}{}{\prime}.\mp@subsup{a}{}{\prime}\in(\mp@subsup{\mathcal{R}}{+}{}\Psiv)\wedge\mp@subsup{a}{}{\prime}\not=a})
using nb
by fastforce
then show ?thesis
proof (cases)
case A1
then have ?t (v,a)= Some True
unfolding map-add-dom-app-simps(1)[OF True]
using nb (1)
by fast
moreover have ?t' (v,a)=Some True
using n\mp@subsup{b}{12}{[}[OF A1].
ultimately show ?thesis..
next
case A2
then have ?t (v,a)= Some False
unfolding map-add-dom-app-simps(1)[OF True]
using nb9(2)
by blast
moreover have ?t' (v,a)=Some False
using n\mp@subsup{b}{13}{}[OF True A2].
ultimately show ?thesis..
qed
next
case False
moreover have ?t (v,a)=Some (the (sv)=a)
using nb }\mp@subsup{1}{14}{[OF v-a-in-dom-t False].
moreover have ?t'}(v,a)=\mathrm{ Some (the (s v)=a)
using nb 15 [OF v-a-in-dom-t False].
ultimately show ?thesis
by argo
qed
} note n\mp@subsup{b}{18}{}= this

```
```

moreover {
fix }v
assume (v,a)\indom? ?t'
hence ?t (v,a)=? ?t' (v,a)
using n\mp@subsup{b}{17}{}n\mp@subsup{b}{18}{}
by presburger
}
- TODO slow.
ultimately have ?t }\mp@subsup{\subseteq}{m}{}\mathrm{ ? ?'t' and ? 't'}\mp@subsup{\subseteq}{m}{}\mathrm{ ? ?
unfolding map-le-def
by fastforce+
thus ?thesis
using map-le-antisym[of ?t ?t']
by fast
qed

```
- NOTE This is the essential step in the SAS+/STRIPS equivalence theorem. We show that executing a given parallel STRIPS operator ops \({ }^{\prime}\) on the corresponding STRIPS state \(s^{\prime}=\varphi_{S} \Psi s\) yields the same state as executing the transformed SAS+ parallel operator ops \(=\left[\varphi_{O}^{-1}(\varphi \Psi) o p^{\prime} . o p^{\prime} \leftarrow o p s^{\prime}\right]\) on the original SAS+ state \(s\) and the transforming the resultant SAS+ state to its corresponding STRIPS state.
lemma sas-plus-equivalent-to-strips-i-a-XII:
assumes is-valid-problem-sas-plus \(\Psi\)
and \(\forall o p^{\prime} \in\) set ops \({ }^{\prime} . o p^{\prime} \in \operatorname{set}\left((\varphi \Psi)_{\mathcal{O}}\right)\)
shows execute-parallel-operator \(\left(\varphi_{S} \Psi s\right)\) ops \({ }^{\prime}\)
\(=\varphi_{S} \Psi\) (execute-parallel-operator-sas-plus \(\left.s\left[\varphi_{O}^{-1} \Psi o p^{\prime} . o p^{\prime} \leftarrow o p s\right\}\right)\)
using assms
proof (induction ops \({ }^{\prime}\) arbitrary: s)
case Nil
then show ?case
unfolding execute-parallel-operator-def execute-parallel-operator-sas-plus-def
by \(\operatorname{simp}\)
next
case (Cons op' ops \({ }^{\prime}\) )
let ? \(\Pi=\varphi \Psi\)
let \(? t^{\prime}=\left(\varphi_{S} \Psi s\right)++\) map-of (effect-to-assignments op \(\left.{ }^{\prime}\right)\)
and \(? t=s++\) map-of \(\left(\right.\) effect-of \(\left.\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right)\right)\)
have \(n b_{1}: ? t^{\prime}=\varphi_{S} \Psi ? t\)
using sas-plus-equivalent-to-strips-i-a-XI[OF assms(1)] Cons.prems(2)
by force
\{
have \(\forall o p^{\prime} \in\) set ops \({ }^{\prime}\). op \({ }^{\prime} \in\) set (strips-problem.operators-of ? \(\Pi\) )
using Cons.prems(2)
by simp
then have execute-parallel-operator \(\left(\varphi_{S} \Psi\right.\) ?t) ops \({ }^{\prime}\)
\(=\varphi_{S} \Psi\) (execute-parallel-operator-sas-plus ? \(\left.\left[\varphi_{O}{ }^{-1} \Psi x . x \leftarrow o p s\right]\right)\)
using Cons.IH[OF Cons.prems(1), of ?t]
by fastforce
```

    hence execute-parallel-operator ?t' ops'
        = \varphi
        using nb
        by argo
    }
    thus ?case
    by simp
    qed
lemma sas-plus-equivalent-to-strips-i-a-XIII:
assumes is-valid-problem-sas-plus \Psi
and }\forallo\mp@subsup{p}{}{\prime}\in\mathrm{ set ops'.op'}\in\operatorname{set}((\varphi\Psi\mp@subsup{)}{\mathcal{O}}{}
and (\mp@subsup{\varphi}{S}{}\PsiG)\subseteqm}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan
(execute-parallel-operator ( }\mp@subsup{\varphi}{S}{\PsiI) ops')}
shows }(\mp@subsup{\varphi}{S}{}\PsiG)\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan
( }\mp@subsup{\varphi}{S}{}\Psi(\mathrm{ execute-parallel-operator-sas-plus I [ }\mp@subsup{\varphi}{O}{-1}\Psi op'.op'\leftarrowops`]))
proof -
let ?I' = ( }\mp@subsup{\varphi}{S}{}\PsiI
and ?G' }=\mp@subsup{\varphi}{S}{}\Psi
and ?ops = [\mp@subsup{\varphi}{O}{-1}\Psio\mp@subsup{p}{}{\prime}.o\mp@subsup{p}{}{\prime}}\leftarrowop\mp@subsup{s}{}{\prime}
and ?\Pi=\varphi\Psi
let ?J = execute-parallel-operator-sas-plus I ?ops
{
fix va
assume (v,a)\indom ? G'
then have ?G'}(v,a)=\mathrm{ execute-parallel-plan
(execute-parallel-operator ?I' ops')}\pi(v,a
using assms(3)
unfolding map-le-def
by auto
hence ?G'}(v,a)=\mathrm{ execute-parallel-plan ( }\mp@subsup{\varphi}{S}{}\Psi ?J) \pi (v,a
using sas-plus-equivalent-to-strips-i-a-XII[OF assms(1, 2)]
by simp
}
thus ?thesis
unfolding map-le-def
by fast
qed

```
- NOTE This is a more abstract formulation of the proposition in sas-plus-equivalent-to-strips-i which is better suited for induction proofs. We essentially claim that given a plan the execution in STRIPS semantics of which solves the problem of reaching a transformed goal state \(\varphi_{S} \Psi G\) from a transformed initial state \(\varphi_{S} \Psi I\)-such as the goal and initial state of an induced STRIPS problem for a SAS+ problem-is equivalent to an execution in SAS + semantics of the transformed plan \(\varphi_{P}^{-1}(\varphi \Psi) \pi\) w.r.t to the original initial state \(I\) and original goal state \(G\).
lemma sas-plus-equivalent-to-strips-i-a:
assumes is-valid-problem-sas-plus \(\Psi\)
and \(\operatorname{dom} I \subseteq \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
```

    and }\forallv\indom I. the (Iv)\in\mp@subsup{\mathcal{R}}{+}{}\Psi
    and dom G\subseteqset ((\Psi) (\mathcal{V})
    and }\forallv\indomG\mathrm{ . the (Gv) G (्र्}+\Psi
    and }\forallop\mp@subsup{s}{}{\prime}\in\mathrm{ set }\pi.\forallo\mp@subsup{p}{}{\prime}\in\mathrm{ set ops'.op' }\in\operatorname{set}((\varphi\Psi\mp@subsup{)}{\mathcal{O}}{\prime}
    and (\mp@subsup{\varphi}{S}{}\PsiG)\subseteq\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan ( }\mp@subsup{\varphi}{S}{\Psi}\Psi|)\pi
    shows G}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan-sas-plus I ( }\mp@subsup{\varphi}{P}{}\mp@subsup{}{}{-1}\Psi\pi
    proof -
let ?vs = variables-of }
and ?\psi = \mp@subsup{\varphi}{P}{-1}\Psi\pi
show ?thesis
using assms
proof (induction \pi arbitrary:I)
case Nil
then have ( }\mp@subsup{\varphi}{S}{\Psi}\PsiG)\mp@subsup{\subseteq}{m}{}(\mp@subsup{\varphi}{S}{}\PsiI
by fastforce
then have G\subseteq\subseteqm
using state-to-strips-state-map-le-iff[OF assms(1, 4, 5)]
by blast
thus ?case
unfolding SAS-Plus-STRIPS.strips-parallel-plan-to-sas-plus-parallel-plan-def
strips-parallel-plan-to-sas-plus-parallel-plan-def
by fastforce
next
case (Cons ops'}\pi\mathrm{ )
let ?D = range-of }
and ?\Pi = \varphi\Psi
and ?I' = \mp@subsup{\varphi}{S}{}\PsiI
and ?}\mp@subsup{G}{}{\prime}=\mp@subsup{\varphi}{S}{}\Psi
let ?ops = [ }\mp@subsup{\varphi}{O}{-1}\Psi \Psio\mp@subsup{p}{}{\prime}.o\mp@subsup{p}{}{\prime}\leftarrowops'
let ?J = execute-parallel-operator-sas-plus I ?ops
and ?'J' = execute-parallel-operator ? I' ops'
have n\mp@subsup{b}{1}{}: set ops'\subseteq set ((?\Pi\mp@subsup{)}{\mathcal{O}}{\prime})
using Cons.prems(6)
unfolding STRIPS-Semantics.is-parallel-solution-for-problem-def list-all-iff
ListMem-iff
by fastforce
{
fix op
assume op \in set ?ops
moreover obtain op' where o\mp@subsup{p}{}{\prime}\in\mathrm{ set ops' and op = 的 -1 }\Psio\mp@subsup{p}{}{\prime}
using calculation
by auto
moreover have op'\in set ((?\Pi)
using nb b calculation(2)
by blast
moreover obtain o\mp@subsup{p}{}{\prime\prime}}\mathrm{ where }o\mp@subsup{p}{}{\prime\prime}\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O+}}{})\mathrm{ and }o\mp@subsup{p}{}{\prime}=\mp@subsup{\varphi}{O}{}\Psi \Psi\mp@subsup{p}{}{\prime\prime
using calculation(4)
by auto
moreover have op =op"

```
using sas-plus-operator-inverse-is[OF assms(1) calculation(5)] calcula\(\operatorname{tion}(3,6)\)
by presburger
ultimately have \(o p \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right) \wedge\left(\exists o p^{\prime} \in \operatorname{set} o p s^{\prime} . o p^{\prime}=\varphi_{O} \Psi o p\right)\)
by blast
\(\}\) note \(n b_{2}=\) this
\{
fix op \(v a\)
assume \(o p \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)and \((v, a) \in \operatorname{set}(\) effect-of op)
moreover have \(o p \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)
using \(n b_{2}\) calculation(1)
by blast
moreover have is-valid-operator-sas-plus \(\Psi\) op
using is-valid-problem-sas-plus-then(2) Cons.prems(1) calculation(3)
by blast
ultimately have \(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
using is-valid-operator-sas-plus-then(3)
by fastforce
\(\}\) note \(n b_{3}=\) this
\{
fix \(o p\)
assume op \(\in\) set ?ops
then have \(o p \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)
using \(n b_{2}\)
by blast
then have is-valid-operator-sas-plus \(\Psi\) op
using is-valid-problem-sas-plus-then(2) Cons.prems(1)
by blast
hence \(\forall(v, a) \in \operatorname{set}\left(\right.\) effect-of op). \(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
\(\wedge a \in \mathcal{R}_{+} \Psi v\)
using is-valid-operator-sas-plus-then (3,4)
by fast
\} note \(n b_{4}=\) this
show ?case
proof (cases STRIPS-Semantics.are-all-operators-applicable ? \(I^{\prime}\) ops \({ }^{\prime}\)
\(\wedge\) STRIPS-Semantics.are-all-operator-effects-consistent ops')
case True
\{
have \(\operatorname{dom} I \subseteq \operatorname{set}\left((\Psi)_{\mathcal{V}+}\right)\)
using Cons.prems(2)
by blast
hence \(\left(\varphi_{S}{ }^{-1} \Psi ? I^{\prime}\right)=I\)
using strips-state-to-state-inverse-is \([O F\)
Cons.prems(1) - Cons.prems(3)]
by argo
\}
then have are-all-operators-applicable-in I ?ops
\(\wedge\) are-all-operator-effects-consistent ?ops
using sas-plus-equivalent-to-strips-i-a-IV[OF assms(1) nb \(b_{1}\), of I] True by \(\operatorname{simp}\)
moreover have \(\left(\varphi_{P}{ }^{-1} \Psi\left(\right.\right.\) ops \(\left.\left.^{\prime} \# \pi\right)\right)=\) ? ops \(\#\left(\varphi_{P}{ }^{-1} \Psi \pi\right)\)
unfolding SAS-Plus-STRIPS.strips-parallel-plan-to-sas-plus-parallel-plan-def
strips-parallel-plan-to-sas-plus-parallel-plan-def
SAS-Plus-STRIPS.strips-op-to-sasp-def
strips-op-to-sasp-def
by \(\operatorname{simp}\)
ultimately have execute-parallel-plan-sas-plus \(I\left(\varphi_{P}{ }^{-1} \Psi\left(\right.\right.\) ops \(\left.\left.^{\prime} \# \pi\right)\right)\)
\(=\) execute-parallel-plan-sas-plus ? \(J\left(\varphi_{P}{ }^{-1} \Psi \pi\right)\)
by force
\(\}\) note \(n b_{5}=t h i s\)
- Show the goal using the IH.
\{
have dom-J-subset-eq-vs: dom ? \(J \subseteq\) set \(\left((\Psi)_{\mathcal{V}+}\right)\)
using sas-plus-equivalent-to-strips-i-a-IX[OF Cons.prems(2)] \(n b_{2} n b_{4}\)
by blast
moreover \{
have set \(\left((\Psi)_{\mathcal{V}_{+}}\right) \subseteq\) dom (range-of \(\Psi\) )
using is-valid-problem-sas-plus-then(1)[OF assms(1)]
by fastforce
moreover have \(\forall v \in \operatorname{dom} I\). the ( \(I v) \in\) set (the (range-of \(\Psi v\) ))
using Cons.prems(2, 3) assms(1) set-the-range-of-is-range-of-sas-plus-if
by force
moreover have \(\forall o p \in\) set ?ops. \(\forall(v, a) \in\) set (effect-of op).
\(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right) \wedge a \in \operatorname{set}(\) the \((? D v))\)
using set-the-range-of-is-range-of-sas-plus-if assms(1) \(n b_{4}\)
by fastforce
moreover have \(v\)-in-dom-J-range: \(\forall v \in\) dom ? J. the (? J v) \(\in\) set (the
\((? D v))\)
using sas-plus-equivalent-to-strips-i-a-X[of
I set \(\left((\Psi)_{\mathcal{V}_{+}}\right)\)? \(D\) ?ops, OF Cons.prems(2)] calculation(1, 2, 3)
by fastforce
\{
fix \(v\)
assume \(v \in \operatorname{dom}\) ?J
moreover have \(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
using \(n b_{2}\) calculation dom-J-subset-eq-vs
by blast
moreover have set (the (range-of \(\Psi v))=\mathcal{R}_{+} \Psi v\)
using set-the-range-of-is-range-of-sas-plus-if[OF assms(1)]
calculation(2)
by presburger
ultimately have the \((? J v) \in \mathcal{R}_{+} \Psi v\)
using \(n b_{3} v\)-in-dom-J-range
by blast
\}
ultimately have \(\forall v \in\) dom ? J. the (?J \(v) \in \mathcal{R}_{+} \Psi v\)
```

            by fast
        }
        moreover have }\forallop\mp@subsup{s}{}{\prime}\in\mathrm{ set }\pi.\forallo\mp@subsup{p}{}{\prime}\in\mathrm{ set ops'.op ' }\in\mathrm{ set (( }\varphi\Psi\mp@subsup{)}{\mathcal{O}}{\prime}
        using Cons.prems(6)
        by simp
    moreover {
        have ?G' }\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan ?J' }
            using Cons.prems(7) True
            by auto
    hence ( }\mp@subsup{\varphi}{S}{\Psi \PsiG)\subseteqm}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan ( }\mp@subsup{\varphi}{S}{\Psi}\Psi\mathrm{ ?J) }
        using sas-plus-equivalent-to-strips-i-a-XIII[OF Cons.prems(1)] nb b
        by blast
    }
    ultimately have G\subseteqm execute-parallel-plan-sas-plus I (\mp@subsup{\varphi}{P}{-1}\Psi(ops'
    
# \pi))

            using Cons.IH[of ?J, OF Cons.prems(1) - - Cons.prems(4, 5)]
    Cons.prems(6) nb 5
by presburger
}
thus ?thesis.
next
case False
then have ? G' }\mp@subsup{\subseteq}{m}{m}\mathrm{ ? I'
using Cons.prems(7)
by force
moreover {
have dom I\subseteq set ?vs
using Cons.prems(2)
by simp
hence }\neg\mathrm{ (are-all-operators-applicable-in I ?ops
^ are-all-operator-effects-consistent ?ops)
using sas-plus-equivalent-to-strips-i-a-VIII[OF Cons.prems(1) -
Cons.prems(3) n\mp@subsup{b}{1}{}]
False
by force
}
moreover {
have ( }\mp@subsup{\varphi}{P}{}\mp@subsup{}{}{-1}\Psi(op\mp@subsup{s}{}{\prime}\#\pi))=\mathrm{ ?ops \# ( }\mp@subsup{\varphi}{P}{-1}\Psi\pi
unfolding SAS-Plus-STRIPS.strips-parallel-plan-to-sas-plus-parallel-plan-def
strips-parallel-plan-to-sas-plus-parallel-plan-def
SAS-Plus-STRIPS.strips-op-to-sasp-def
strips-op-to-sasp-def
by simp
hence G}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan-sas-plus I (?ops \# ( }\mp@subsup{\varphi}{P}{}\mp@subsup{}{}{-1}\Psi\pi)
\longleftrightarrowG\subseteqm
using calculation(2)
by force
}
ultimately show ?thesis

```
```

            using state-to-strips-state-map-le-iff[OF Cons.prems(1, 4, 5)]
            unfolding SAS-Plus-STRIPS.strips-parallel-plan-to-sas-plus-parallel-plan-def
                strips-parallel-plan-to-sas-plus-parallel-plan-def
                SAS-Plus-STRIPS.strips-op-to-sasp-def
                strips-op-to-sasp-def
                    by force
            qed
    qed
    qed

```
- NOTE Show that a solution for the induced STRIPS problem for the given valid SAS+ problem, corresponds to a solution for the given SAS+ problem.
Note that in the context of the SAS+ problem solving pipeline, we
1. convert the given valid SAS \(+\Psi\) problem to the corresponding STRIPS problem \(\Pi\) (this is implicitely also valid by lemma is-valid-problem-sas-plus-then-strips-transformation-too); then,
2. get a solution \(\pi\)-if it exists-for the induced STRIPS problem by executing SATPlan; and finally,
3. convert \(\pi\) back to a solution \(\psi\) for the SAS + problem.
```

lemma sas-plus-equivalent-to-strips-i:
assumes is-valid-problem-sas-plus $\Psi$
and STRIPS-Semantics.is-parallel-solution-for-problem
$(\varphi \Psi) \pi$
shows goal-of $\Psi \subseteq_{m}$ execute-parallel-plan-sas-plus
(sas-plus-problem.initial-of $\Psi$ ) $\left(\varphi_{P}{ }^{-1} \Psi \pi\right)$
proof -
let $? v s=$ variables-of $\Psi$
and $? I=$ initial-of $\Psi$
and $? G=$ goal-of $\Psi$
let $? \Pi=\varphi \Psi$
let ? $G^{\prime}=$ strips-problem.goal-of ? $\Pi$
and $? I^{\prime}=$ strips-problem.initial-of ? $\Pi$
let $? \psi=\varphi_{P}{ }^{-1} \Psi \pi$
have $d o m ? I \subseteq$ set ?vs
using is-valid-problem-sas-plus-then(3) assms(1)
by auto
moreover have $\forall v \in d o m$ ?I. the (?I $v) \in \mathcal{R}_{+} \Psi v$
using is-valid-problem-sas-plus-then(4) assms(1) calculation
by auto
moreover have dom ? $G \subseteq$ set ? vs and $\forall v \in \operatorname{dom}$ ? $G$. the $(? G v) \in \mathcal{R}_{+} \Psi v$
using is-valid-problem-sas-plus-then $(5,6)$ assms (1)
by blast+
moreover have $\forall o p s^{\prime} \in$ set $\pi$. $\forall o p^{\prime} \in$ set ops ${ }^{\prime}$. op ${ }^{\prime} \in \operatorname{set}\left((? \Pi)_{\mathcal{O}}\right)$
using is-parallel-solution-for-problem-operator-set[OF assms(2)]
by $\operatorname{simp}$
moreover \{
have ? $G^{\prime} \subseteq_{m}$ execute-parallel-plan ? $I^{\prime} \pi$

```
```

        using assms(2)
        unfolding STRIPS-Semantics.is-parallel-solution-for-problem-def..
    moreover have ?'G}\mp@subsup{G}{}{\prime}=\mp@subsup{\varphi}{S}{}\Psi?G\mathrm{ and ? I' = 的 }\Psi?
        by simp+
    ultimately have ( }\mp@subsup{\varphi}{S}{}\Psi??G)\mp@subsup{\preceq}{m}{}\mathrm{ execute-parallel-plan ( }\mp@subsup{\varphi}{S}{}\Psi?I)
        by simp
    }
    ultimately show?thesis
    using sas-plus-equivalent-to-strips-i-a[OF assms(1)]
    by simp
    qed

```
- NOTE Show that the operators for a given solution \(\pi\) to the induced STRIPS problem for a given SAS+ problem correspond to operators of the SAS+ problem.
lemma sas-plus-equivalent-to-strips-ii:
    assumes is-valid-problem-sas-plus \(\Psi\)
    and STRIPS-Semantics.is-parallel-solution-for-problem \((\varphi \Psi) \pi\)
    shows list-all (list-all ( \(\lambda\) op. ListMem op (operators-of \(\Psi)\) )) \(\left(\varphi_{P}{ }^{-1} \Psi \pi\right)\)
proof -
    let \(? \Pi=\varphi \Psi\)
    let ?ops \(=\) operators-of \(\Psi\)
        and \(? \psi=\varphi_{P}{ }^{-1} \Psi \pi\)
    have is-valid-problem-strips? \(?\)
        using is-valid-problem-sas-plus-then-strips-transformation-too[OF assms(1)]
        by \(\operatorname{simp}\)
    have \(n b_{1}: \forall o p^{\prime} \in \operatorname{set}\left((? \Pi)_{\mathcal{O}}\right) .\left(\exists\right.\) op \(\in\) set ?ops. op \({ }^{\prime}=\left(\varphi_{O} \Psi\right.\) op \(\left.)\right)\)
        by auto
    \{
        fix ops' op' op
        assume ops \({ }^{\prime} \in\) set \(\pi\) and \(o p^{\prime} \in\) set ops \({ }^{\prime}\)
        then have op \({ }^{\prime} \in\) set (strips-problem.operators-of ? \(\Pi\) )
            using is-parallel-solution-for-problem-operator-set[OF assms(2)]
            by simp
        then obtain \(o p\) where \(o p \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)and \(o p^{\prime}=\left(\varphi_{O} \Psi o p\right)\)
            by auto
        then have \(\left(\varphi_{O}{ }^{-1} \Psi o p^{\prime}\right) \in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)
            using sas-plus-operator-inverse-is[OF assms(1)]
            by presburger
    \}
    thus ?thesis
        unfolding list-all-iff ListMem-iff
            strips-parallel-plan-to-sas-plus-parallel-plan-def
            SAS-Plus-STRIPS.strips-parallel-plan-to-sas-plus-parallel-plan-def
            SAS-Plus-STRIPS.strips-op-to-sasp-def
            strips-op-to-sasp-def
        by auto
qed

We now show that for a parallel solution \(\pi\) of \(\Pi\) the \(\mathrm{SAS}+\) plan \(\psi \equiv \varphi_{P}{ }^{-1}\)
\(\Psi \pi\) yielded by the STRIPS to SAS + plan transformation is a solution for \(\Psi\). The proof uses the definition of parallel STRIPS solutions and shows that the execution of \(\psi\) on the initial state of the SAS+ problem yields a state satisfying the problem's goal state, i.e.
```

G\subseteqm}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan-sas-plus I }

```
and by showing that all operators in all parallel operators of \(\psi\) are operators of the problem.
```

theorem
sas-plus-equivalent-to-strips:
assumes is-valid-problem-sas-plus $\Psi$
and STRIPS-Semantics.is-parallel-solution-for-problem $(\varphi \Psi) \pi$
shows is-parallel-solution-for-problem $\Psi\left(\varphi_{P}{ }^{-1} \Psi \pi\right)$
proof -
let ? $I=$ initial-of $\Psi$
and ? $G=$ goal-of $\Psi$
and ?ops $=$ operators-of $\Psi$
and $? \psi=\varphi_{P}{ }^{-1} \Psi \pi$
show ?thesis
unfolding is-parallel-solution-for-problem-def Let-def
proof (rule conjI)
show ? $G \subseteq_{m}$ execute-parallel-plan-sas-plus ?I ? $\psi$
using sas-plus-equivalent-to-strips-i [OF assms].
next
show list-all (list-all ( $\lambda$ op. ListMem op ?ops)) ? $\psi$
using sas-plus-equivalent-to-strips-ii[OF assms].
qed
qed
private lemma strips-equivalent-to-sas-plus-i-a-I:
assumes is-valid-problem-sas-plus $\Psi$
and $\forall o p \in$ set ops. op $\in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)$
and $o p^{\prime} \in \operatorname{set}\left[\varphi_{O} \Psi\right.$ op. op $\left.\leftarrow o p s\right]$
obtains $o p$ where $o p \in$ set ops
and $o p^{\prime}=\varphi_{O} \Psi o p$
proof -
let $? \Pi=\varphi \Psi$
let ?ops $=$ operators-of $\Psi$
obtain $o p$ where $o p \in$ set ops and $o p^{\prime}=\varphi_{O} \Psi o p$
using assms(3)
by auto
thus ?thesis
using that
by blast
qed
private corollary strips-equivalent-to-sas-plus-i-a-II:
assumesis-valid-problem-sas-plus $\Psi$

```
```

    and }\forallop\in\mathrm{ set ops.op }\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O+}}{}
    and o\mp@subsup{p}{}{\prime}\in\operatorname{set}[\mp@subsup{\varphi}{O}{}\Psiop.op\leftarrowops]
    shows op'\in set ((\varphi\Psi)\mathcal{O})
    and is-valid-operator-strips ( }\varphi\Psi)o\mp@subsup{)}{}{\prime
    proof -
let ?\Pi=\varphi\Psi
let ?ops = operators-of }
and ?ops' = strips-problem.operators-of ?\Pi
obtain op where op-in:op\in set ops and op'-is:op' = \varphiO\Psi op
using strips-equivalent-to-sas-plus-i-a-I[OF assms].
then have nb:op' \in set ((\varphi\Psi\mp@subsup{)}{\mathcal{O}}{)}
using assms(2) op-in op'-is
by fastforce
thus o\mp@subsup{p}{}{\prime}\in\operatorname{set}((\varphi\Psi\mp@subsup{)}{\mathcal{O}}{)})
and is-valid-operator-strips?\Pi op'
proof -
have \forallop'\in set ?ops'. is-valid-operator-strips ?\Pi op'
using is-valid-problem-sas-plus-then-strips-transformation-too-iii[[OF assms(1)]
unfolding list-all-iff.
thus is-valid-operator-strips ?\Pi op'
using nb
by fastforce
qed fastforce
qed
lemma strips-equivalent-to-sas-plus-i-a-III:
assumes is-valid-problem-sas-plus \Psi
and}\forallop\in set ops.op\in set ((\Psi\mp@subsup{)}{\mathcal{O+}}{}
shows execute-parallel-operator ( ( }\mp@subsup{\}{S}{\Psi
=( }\mp@subsup{\varphi}{S}{}\Psi(\mathrm{ execute-parallel-operator-sas-plus s ops)}
proof -
{
fix op s
assume op }\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O+}}{}
moreover have (}(\mp@subsup{\varphi}{O}{}\Psiop)\in\operatorname{set}((\varphi\Psi\mp@subsup{)}{\mathcal{O}}{}
using calculation
by simp
moreover have ( }\mp@subsup{\varphi}{S}{}\Psis)++\mathrm{ map-of (effect-to-assignments ( }\mp@subsup{\varphi}{O}{}
=( }\mp@subsup{\varphi}{S}{}\Psi(s++\mathrm{ map-of (effect-of ( }\mp@subsup{\varphi}{O}{-1}\Psi(\mp@subsup{\varphi}{O}{}\Psi \Psiop))))
using sas-plus-equivalent-to-strips-i-a-XI[OF assms(1) calculation(2)]
by blast
moreover have ( }\mp@subsup{\varphi}{O}{-1}\Psi(\mp@subsup{\varphi}{O}{}\Psiop))=o
using sas-plus-operator-inverse-is[OF assms(1) calculation(1)].
ultimately have (\mp@subsup{\varphi}{S}{}\Psis)>>(\mp@subsup{\varphi}{O}{}\Psiop)
=(\mp@subsup{\varphi}{S}{}\Psi(s>>+op))
unfolding execute-operator-def execute-operator-sas-plus-def
by simp
} note n\mp@subsup{b}{1}{}=this

```
```

show ?thesis
using assms
proof (induction ops arbitrary: s)
case Nil
then show ?case
unfolding execute-parallel-operator-def execute-parallel-operator-sas-plus-def
by simp
next
case (Cons op ops)
let ?t =s>>+op
let ? s' = \varphi S \Psi s
and ?ops' = [ [ O \Psi op.op \leftarrowop \#ops]
let ? }\mp@subsup{t}{}{\prime}=?,\mp@subsup{s}{}{\prime}>>(\mp@subsup{\varphi}{O}{}\Psiop
have execute-parallel-operator ?s' ?ops'
= execute-parallel-operator ? t' [ [ }\mp@subsup{\varphi}{O}{}\Psix.x\leftarrowops
unfolding execute-operator-def
by simp
moreover have ( }\mp@subsup{\varphi}{S}{}\Psi\mathrm{ (execute-parallel-operator-sas-plus s (op \# ops)))
=(\mp@subsup{\varphi}{S}{}\Psi(execute-parallel-operator-sas-plus ?t ops))
unfolding execute-operator-sas-plus-def
by simp
moreover {
have ? t t = ( }\mp@subsup{\varphi}{S}{}\Psi?t
using n\mp@subsup{b}{1}{}}\mathrm{ Cons.prems(2)
by simp
hence execute-parallel-operator ? t'}[\mp@subsup{\varphi}{O}{}\Psix.x\leftarrowops
=(\mp@subsup{\varphi}{S}{}\Psi(execute-parallel-operator-sas-plus ?t ops))
using Cons.IH[of ?t] Cons.prems
by simp
}
ultimately show ?case
by argo
qed
qed
private lemma strips-equivalent-to-sas-plus-i-a-IV:
assumes is-valid-problem-sas-plus \Psi
and}\forallop\in set ops.op\in set ((\Psi\mp@subsup{)}{\mp@subsup{\mathcal{O}}{+}{*}}{}
and are-all-operators-applicable-in I ops
\wedge ~ a r e - a l l - o p e r a t o r - e f f e c t s - c o n s i s t e n t ~ o p s
shows STRIPS-Semantics.are-all-operators-applicable ( }\mp@subsup{\varphi}{S}{}\PsiI)[\mp@subsup{\varphi}{O}{\Psi}\Psi op.op
ops]
^ STRIPS-Semantics.are-all-operator-effects-consistent [ [ }\mp@subsup{\varphi}{O}{}\Psi\mathrm{ op.op }\leftarrowops
proof -
let ?vs = variables-of \Psi
and ?ops = operators-of }
let ? 'I' = \varphi S \Psi I
and ?ops' }=[\mp@subsup{\varphi}{O}{}\Psi\mathrm{ \$p.op }\leftarrowops

```
```

have n\mp@subsup{b}{1}{}:\forallop \in set ops. is-operator-applicable-in I op
using assms(3)
unfolding are-all-operators-applicable-in-def list-all-iff
by blast
have }n\mp@subsup{b}{2}{}:\forallop\in\mathrm{ set ops. is-valid-operator-sas-plus }\Psi\mathrm{ op
using is-valid-problem-sas-plus-then(2) assms(1, 2)
unfolding is-valid-operator-sas-plus-def
by auto
have n\mp@subsup{b}{3}{}:\forallop\in set ops. map-of (precondition-of op) }\mp@subsup{\subseteq}{m}{}
using n\mp@subsup{b}{1}{}
unfolding is-operator-applicable-in-def list-all-iff
by blast
{
fix }o\mp@subsup{p}{1}{}o\mp@subsup{p}{2}{
assume op, 的 set ops and op 的 set ops
hence are-operator-effects-consistent op (op 年
using assms(3)
unfolding are-all-operator-effects-consistent-def list-all-iff
by blast
} note nb }=\mathrm{ this
{
fix }o\mp@subsup{p}{1}{}O\mp@subsup{p}{2}{
assume op, 的 set ops and op 的 set ops
hence }\forall(v,a)\in\operatorname{set}(effect-of o\mp@subsup{p}{1}{}).\forall(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\operatorname{set}(effect-of o\mp@subsup{p}{2}{})
v\not=\mp@subsup{v}{}{\prime}\veea=\mp@subsup{a}{}{\prime}
using nb
unfolding are-operator-effects-consistent-def Let-def list-all-iff
by presburger
} note nb
{
fix op '' op ''I
assume op (' '\in set ?ops'
and op, ''\in set ?ops'
and }\exists(v,a)\in\operatorname{set (add-effects-of op ('). \exists ( }\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\operatorname{set}(delete-effects-of op 2')
(v,a)=(v
moreover obtain op op op
where o\mp@subsup{p}{1}{}\in set ops
and}o\mp@subsup{p}{1}{\prime}\mp@subsup{}{}{\prime}=\mp@subsup{\varphi}{O}{}\Psi\$o\mp@subsup{p}{1}{
and op
and }o\mp@subsup{p}{2}{\prime}\mp@subsup{}{}{\prime}=\mp@subsup{\varphi}{O}{}\Psi\quad\Psi\mp@subsup{p}{2}{
using strips-equivalent-to-sas-plus-i-a-I[OF assms(1, 2)] calculation(1, 2)
by auto
moreover have is-valid-operator-sas-plus \Psi op
and is-valid-operator-op 2: is-valid-operator-sas-plus \Psi op 2
using calculation(4,6) nb
by blast+
moreover obtain v v
where (v,a) \in set (add-effects-of op (}\mp@subsup{}{1}{\prime}
and (v', a')\in set (delete-effects-of op ' ')

```
```

            and \((v, a)=\left(v^{\prime}, a^{\prime}\right)\)
            using calculation
            by blast
    moreover have \((v, a) \in \operatorname{set}\) (effect-of op \(p_{1}\) )
    using calculation \((5,10)\)
    unfolding SAS-Plus-STRIPS.sasp-op-to-strips-def
        sasp-op-to-strips-def Let-def
    by fastforce
    moreover have \(v=v^{\prime}\) and \(a=a^{\prime}\)
        using calculation(12)
        by \(\operatorname{simp}+\)
    moreover \{
    have \(\left(v^{\prime}, a^{\prime}\right) \in\left(\bigcup(v, a) \in \operatorname{set}\left(\right.\right.\) effect-of op \(\left.{ }_{2}\right)\).
        \(\left.\left\{\left(v, a^{\prime}\right) \mid a^{\prime} . a^{\prime} \in\left(\mathcal{R}_{+} \Psi v\right) \wedge a^{\prime} \neq a\right\}\right)\)
        using sasp-op-to-strips-set-delete-effects-is
            calculation (7, 9, 11)
        by blast
    then obtain \(v^{\prime \prime} a^{\prime \prime}\) where \(\left(v^{\prime \prime}, a^{\prime \prime}\right) \in \operatorname{set}\left(\right.\) effect-of op \({ }_{2}\) )
        and \(\left(v^{\prime}, a^{\prime}\right) \in\left\{\left(v^{\prime \prime}, a^{\prime \prime \prime}\right) \mid a^{\prime \prime \prime} \cdot a^{\prime \prime \prime} \in\left(\mathcal{R}_{+} \Psi v^{\prime \prime}\right) \wedge a^{\prime \prime \prime} \neq a^{\prime \prime}\right\}\)
        by blast
    moreover have \(\left(v^{\prime}, a^{\prime \prime}\right) \in \operatorname{set}\left(\right.\) effect-of \(\left.o p_{2}\right)\)
        using calculation
        by blast
    moreover have \(a^{\prime} \in \mathcal{R}_{+} \Psi v^{\prime \prime}\) and \(a^{\prime} \neq a^{\prime \prime}\)
        using calculation(1, 2)
        by fast+
    ultimately have \(\exists a^{\prime \prime} .\left(v^{\prime}, a^{\prime \prime}\right) \in \operatorname{set}(\) effect-of op 2\() \wedge a^{\prime} \in\left(\mathcal{R}_{+} \Psi v^{\prime}\right)\)
        \(\wedge a^{\prime} \neq a^{\prime \prime}\)
        by blast
    \}
    moreover obtain \(a^{\prime \prime}\) where \(a^{\prime} \in \mathcal{R}_{+} \Psi v^{\prime}\)
        and \(\left(v^{\prime}, a^{\prime \prime}\right) \in \operatorname{set}\left(\right.\) effect-of op \({ }_{2}\) )
        and \(a^{\prime} \neq a^{\prime \prime}\)
        using calculation(16)
        by blast
    moreover have \(\exists(v, a) \in \operatorname{set}\left(\right.\) effect-of op \(\left.p_{1}\right) .\left(\exists\left(v^{\prime}, a^{\prime}\right) \in \operatorname{set}\left(\right.\right.\) effect-of op \(\left.p_{2}\right)\).
    \(\left.v=v^{\prime} \wedge a \neq a^{\prime}\right)\)
    using calculation(13, 14, 15, 17, 18, 19)
    by blast
    - TODO slow.
    ultimately have \(\exists o p_{1} \in\) set ops. \(\exists\) op \(p_{2} \in\) set ops. \(\neg\) are-operator-effects-consistent
    $o p_{1} o p_{2}$
unfolding are-operator-effects-consistent-def list-all-iff
by fastforce
\} note $n b_{6}=$ this
show ?thesis
proof (rule conjI)
\{
fix $o p^{\prime}$

```
```

    assume op' \in set ?ops'
    moreover obtain op where op-in:op\in set ops
    and o\mp@subsup{p}{}{\prime}-is:o\mp@subsup{p}{}{\prime}=\mp@subsup{\varphi}{O}{}\Psiop
    and op'-in:o\mp@subsup{p}{}{\prime}\in\operatorname{set}((\varphi\Psi\mp@subsup{)}{\mathcal{O}}{})
    and is-valid-op':is-valid-operator-strips ( }\varphi\Psi)\mathrm{ )op'
    using strips-equivalent-to-sas-plus-i-a-I[OF assms(1, 2)]
        strips-equivalent-to-sas-plus-i-a-II[OF assms(1, 2)] calculation
    by metis
    moreover have is-valid-op: is-valid-operator-sas-plus \Psi op
using n\mp@subsup{b}{2}{}}\mathrm{ calculation(2)..
{
fix va
assume v-a-in-preconditions':(v,a)\in set (strips-operator.precondition-of
op')
have v-a-in-preconditions:(v,a)\in set (precondition-of op)
using op'-is
unfolding SAS-Plus-STRIPS.sasp-op-to-strips-def
sasp-op-to-strips-def Let-def
using v-a-in-preconditions'
by force
moreover have v\in set ?vs and }a\in\mp@subsup{\mathcal{R}}{+}{}\Psi
using is-valid-operator-sas-plus-then(1,2) is-valid-op calculation(1)
by fastforce+
moreover {
have }\forall(v,a)\in\operatorname{set (precondition-of op).}\forall(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\operatorname{set}\mathrm{ (precondition-of
op).
v\not=\mp@subsup{v}{}{\prime}\veea=\mp@subsup{a}{}{\prime}
using is-valid-operator-sas-plus-then(5) is-valid-op
by fast
hence map-of (precondition-of op) v=Some a
using map-of-constant-assignments-defined-if[OF - v-a-in-preconditions]
by blast
}
moreover have v\indom (map-of (precondition-of op))
using calculation(4)
by blast
moreover have Iv=Some a
using nb3
unfolding map-le-def
using op-in calculation(4,5)
by metis
moreover have (v,a)\indom ?I'
using state-to-strips-state-dom-element-iff[OF assms(1)]
calculation(2, 3, 6)
by simp
ultimately have ?I' (v,a) = Some True
using state-to-strips-state-range-is[OF assms(1)]
by simp
}

```
hence STRIPS-Representation.is-operator-applicable-in? \(I^{\prime}\) op \({ }^{\prime}\) unfolding STRIPS-Representation.is-operator-applicable-in-def Let-def list-all-iff
by fast
\}
thus are-all-operators-applicable ? \(I^{\prime}\) ?ops \({ }^{\prime}\)
unfolding are-all-operators-applicable-def list-all-iff
by blast

\section*{next}
\{
fix \(o p_{1}{ }^{\prime} o p_{2}{ }^{\prime}\)
assume \(o p_{1}{ }^{\prime}\)-in-ops': op \({ }^{\prime} \in\) set ?ops \({ }^{\prime}\) and \(o p_{2}{ }^{\prime}\)-in-ops': op \({ }^{\prime}{ }^{\prime} \in\) set ?ops \({ }^{\prime}\)
have STRIPS-Semantics.are-operator-effects-consistent op \({ }_{1}{ }^{\prime} o p_{2}{ }^{\prime}\)
unfolding STRIPS-Semantics.are-operator-effects-consistent-def Let-def
- TODO proof is symmetrical... refactor into nb.
proof (rule conjI)
show \(\neg\) list-ex \(\left(\lambda x\right.\). list-ex \(((=) x)\left(\right.\) delete-effects-of op \(\left.\left.{ }_{2} '\right)\right)\)
(add-effects-of op \({ }_{1}\) )
proof (rule ccontr)
assume \(\neg \neg\) list-ex \(\left(\lambda v\right.\). list-ex \(((=) v)\left(\right.\) delete-effects-of op \(\left.\left.{ }_{2}{ }^{\prime}\right)\right)\)
(add-effects-of op \({ }_{1}\) )
then have \(\exists(v, a) \in\) set (delete-effects-of op \({ }_{2}\) ).
\(\exists\left(v^{\prime}, a^{\prime}\right) \in \operatorname{set}\left(a d d\right.\)-effects-of op \(\left.{ }_{1}\right) .(v, a)=\left(v^{\prime}, a^{\prime}\right)\)
unfolding list-ex-iff
by fastforce
then obtain \(o p_{1} o p_{2}\) where \(o p_{1} \in\) set ops
and \(o p_{2} \in\) set ops
and \(\neg\) are-operator-effects-consistent \(o p_{1} o p_{2}\)
using \(n b_{6}\left[\right.\) OF op \({ }_{1}{ }^{\prime}\)-in-ops \({ }^{\prime} o p_{2}{ }^{\prime}\)-in-ops \(]\)
by blast
thus False
using \(n b_{4}\)
by blast
qed
next
show \(\neg l i s t-e x\left(\lambda v\right.\). list-ex \(((=) v)\left(\right.\) add-effects-of op \(\left.\left._{2}{ }^{\prime}\right)\right)\) (delete-effects-of
\(\left.o p_{1}{ }^{\prime}\right)\)
proof (rule ccontr)
assume \(\neg \neg l i s t-e x\left(\lambda v\right.\). list-ex \(((=) v)\left(\right.\) add-effects-of op \({ }_{2}\) '))
(delete-effects-of op \({ }_{1}{ }^{\prime}\) )
then have \(\exists(v, a) \in\) set (delete-effects-of op \({ }_{1}\) ). \(\exists\left(v^{\prime}, a^{\prime}\right) \in \operatorname{set}\left(a d d-\right.\) effects-of op \(\left.{ }_{2}{ }^{\prime}\right) .(v, a)=\left(v^{\prime}, a^{\prime}\right)\)
unfolding list-ex-iff
by fastforce
then obtain \(o p_{1} o p_{2}\) where \(o p_{1} \in\) set ops and \(o p_{2} \in\) set ops and \(\neg\) are-operator-effects-consistent \(o p_{1} o p_{2}\) using \(n b_{6}\left[O F o p_{2}{ }^{\prime}\right.\)-in-ops \({ }^{\prime} o p_{1}{ }^{\prime}\)-in-ops \(]\)
```

                    by blast
                    thus False
                    using nb 
                    by blast
                    qed
            qed
        }
        thus STRIPS-Semantics.are-all-operator-effects-consistent ?ops'
        unfolding STRIPS-Semantics.are-all-operator-effects-consistent-def list-all-iff
            by blast
        qed
    qed
private lemma strips-equivalent-to-sas-plus-i-a-V:
assumes is-valid-problem-sas-plus \Psi
and}\forallop\in set ops.op\in set ((\Psi\mp@subsup{)}{\mathcal{O+}}{}
and }\neg\mathrm{ (are-all-operators-applicable-in s ops
\wedge ~ a r e - a l l - o p e r a t o r - e f f e c t s - c o n s i s t e n t ~ o p s )
shows }\neg(STRIPS-Semantics.are-all-operators-applicable ( ( \varphi \Psi s) [ [ O \Psi op.o
\leftarrow o p s ]
\wedge STRIPS-Semantics.are-all-operator-effects-consistent [\varphiO \Psi op.op \leftarrowops])
proof -
let ?vs = variables-of }
and ?ops = operators-of }
let ? s}\mp@subsup{s}{}{\prime}=\mp@subsup{\varphi}{S}{}\Psi
and ?ops' = [ }\mp@subsup{\varphi}{O}{}\Psi\mathrm{ U op.op }\leftarrowops
{
fix op
assume op }\in\mathrm{ set ops
hence \existsop\mp@subsup{p}{}{\prime}\in set?ops'.o\mp@subsup{p}{}{\prime}=\mp@subsup{\varphi}{O}{}\Psi \Psiop
by simp
} note n\mp@subsup{b}{1}{}=this
{
fix op
assume op \in set ops
then have op }\in\operatorname{set}((\Psi\mp@subsup{)}{\mathcal{O+}}{}
using assms(2)
by blast
then have is-valid-operator-sas-plus \Psi op
using is-valid-problem-sas-plus-then(2) assms(1)
unfolding is-valid-operator-sas-plus-def
by auto
hence }\forall(v,a)\in\operatorname{set (precondition-of op).}\forall(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\mathrm{ set (precondition-of op).
v\not=\mp@subsup{v}{}{\prime}\vee
using is-valid-operator-sas-plus-then(5)
unfolding is-valid-operator-sas-plus-def
by fast
} note n\mp@subsup{b}{2}{}= this
{

```
```

consider (A) ᄀare-all-operators-applicable-in s ops
| (B) \negare-all-operator-effects-consistent ops
using assms(3)
by blast
hence }\neg\mathrm{ STRIPS-Semantics.are-all-operators-applicable ?s' ?ops'
\vee \neg S T R I P S - S e m a n t i c s . a r e - a l l - o p e r a t o r - e f f e c t s - c o n s i s t e n t ~ ? o p s ' '
proof (cases)
case A
then obtain op where op-in:op\in set ops
and not-precondition-map-le-s: }\neg(\mathrm{ map-of (precondition-of op) }\subseteqms
using }
unfolding are-all-operators-applicable-in-def list-all-iff
is-operator-applicable-in-def
by blast
then obtain op' where o\mp@subsup{p}{}{\prime}-in:o\mp@subsup{p}{}{\prime}\in\mathrm{ set ?ops' and op'-is:op'}=\mp@subsup{\varphi}{O}{\prime}\Psiop
using n\mp@subsup{b}{1}{}
by blast
have \negare-all-operators-applicable?s' ?ops'
proof (rule ccontr)
assume \negᄀare-all-operators-applicable ?s' ?ops'
then have all-operators-applicable: are-all-operators-applicable ?s' ?ops'
by simp
moreover {
fix }
assume v\indom (map-of (precondition-of op))
moreover obtain a where map-of (precondition-of op) v=Some a
using calculation
by blast
moreover have (v,a)\in set (precondition-of op)
using map-of-SomeD[OF calculation(2)].
moreover have (v,a)\in set (strips-operator.precondition-of op')
using op'-is
unfolding sasp-op-to-strips-def
SAS-Plus-STRIPS.sasp-op-to-strips-def
using calculation(3)
by auto
moreover have ?s' (v,a)=Some True
using all-operators-applicable calculation
unfolding are-all-operators-applicable-def
STRIPS-Representation.is-operator-applicable-in-def
is-operator-applicable-in-def Let-def list-all-iff
using op'-in
by fast
moreover have (v,a)\indom?s'
using calculation(5)
by blast
moreover have (v,a)\in set (precondition-of op)
using op'-is calculation(3)
unfolding sasp-op-to-strips-def Let-def

```
```

            by fastforce
            moreover have v\in set ?vs
            and}a\in\mp@subsup{\mathcal{R}}{+}{}\Psi
            and s v\not= None
            using state-to-strips-state-dom-element-iff[OF assms(1)]
                calculation(6)
            by simp+
            moreover have ? s' (v,a)=Some (the (sv)=a)
                            using state-to-strips-state-range-is[OF
            assms(1) calculation(6)].
            moreover have the (s v)=a
                            using calculation(5, 11)
                            by fastforce
                            moreover have s v=Some a
                            using calculation(12) option.collapse[OF calculation(10)]
            by argo
            moreover have map-of (precondition-of op) v=Some a
                using map-of-constant-assignments-defined-if[OF nb }\mp@subsup{\mp@code{2}}{[}{[OF}\mathrm{ op-in]
    calculation(7)].
ultimately have map-of (precondition-of op) v}=s
by argo
}
then have map-of (precondition-of op) \subseteqms
unfolding map-le-def
by blast
thus False
using not-precondition-map-le-s
by simp
qed
thus ?thesis
by simp
next
case }
{
obtain op (op ov v v' a a'
where op
and op -in:op
and v-a-in: (v,a)\in set (effect-of op ( )
and }\mp@subsup{v}{}{\prime}-\mp@subsup{a}{}{\prime}-in:(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\operatorname{set}(effect-of o\mp@subsup{p}{2}{}
and v-is:v=\mp@subsup{v}{}{\prime}}\mathrm{ and }a-is:a\not=\mp@subsup{a}{}{\prime
using }
unfolding are-all-operator-effects-consistent-def
are-operator-effects-consistent-def list-all-iff Let-def
by blast
moreover obtain o\mp@subsup{p}{1}{\prime}}o\mp@subsup{p}{2}{\prime}\mp@subsup{}{}{\prime}\mathrm{ where op, '}\in\mathrm{ set ?ops' and op, '}=\mp@subsup{\varphi}{O}{}\Psi \Psio\mp@subsup{p}{1}{
and o\mp@subsup{p}{1}{\prime}}\mp@subsup{}{}{\prime}\in\mathrm{ set ?ops' and op }\mp@subsup{}{}{\prime}\mp@subsup{}{}{\prime}-is:o\mp@subsup{p}{2}{\prime}\mp@subsup{}{}{\prime}=\mp@subsup{\varphi}{O}{}\Psi \Psio\mp@subsup{p}{2}{
using n\mp@subsup{b}{1}{}[OF calculation(1)] n\mp@subsup{b}{1}{}[OF calculation(2)]
by blast
moreover have (v,a)\in set (add-effects-of op (')

```
```

            using calculation(3, 8)
            unfolding SAS-Plus-STRIPS.sasp-op-to-strips-def
                sasp-op-to-strips-def Let-def
                by force
            moreover {
                have is-valid-operator-sas-plus \Psi op 1
            using assms(2) calculation(1) is-valid-problem-sas-plus-then(2) assms(1)
            unfolding is-valid-operator-sas-plus-def
            by auto
            moreover have is-valid-operator-sas-plus \Psi op 2
                using sublocale-sas-plus-finite-domain-representation-ii(2)[
                    OF assms(1)] assms(2) op 2-in
            by blast
            moreover have a\in\mathcal{R}
            using is-valid-operator-sas-plus-then(4) calculation v-a-in
            unfolding is-valid-operator-sas-plus-def
            by fastforce
            ultimately have (v,a)\in set (delete-effects-of op 2')
            using sasp-op-to-strips-set-delete-effects-is[of \Psi op 2]
                v}\mp@subsup{v}{}{\prime}-\mp@subsup{a}{}{\prime}-in v-is a-i
            using op ''-is
            by blast
            }
            - TODO slow.
            ultimately have }\existso\mp@subsup{p}{1}{\prime}\mp@subsup{}{}{\prime}\in\mathrm{ set ?ops'. }\exists\textrm{op}\mp@subsup{2}{2}{\prime}\in\mathrm{ set ?ops'.
            \exists(v,a) set (delete-effects-of op 2}\mp@subsup{}{}{\prime}).\exists(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime})\in\operatorname{set}(add-effects-of o\mp@subsup{p}{1}{\prime})
            (v,a) = (v', a')
            by fastforce
        }
            then have }\neg\mathrm{ STRIPS-Semantics.are-all-operator-effects-consistent ?ops'
            unfolding STRIPS-Semantics.are-all-operator-effects-consistent-def
                STRIPS-Semantics.are-operator-effects-consistent-def list-all-iff list-ex-iff
    Let-def
by blast
thus ?thesis
by simp
qed
}
thus ?thesis
by blast
qed
lemma strips-equivalent-to-sas-plus-i-a:
assumes is-valid-problem-sas-plus $\Psi$
and dom I\subseteqset ((\Psi)}\mp@subsup{\mathcal{V}}{+}{}

```

```

    and dom G\subseteq set ((\Psi)}\mp@subsup{\mathcal{V}}{+}{\prime}
    and }\forallv\in\operatorname{dom}G\mathrm{ . the (Gv) G 敢 }\Psi
    ```
```

    and }\forallops\in set \psi.\forallop\in set ops.op \in set ((\Psi) () O+ )
    and G}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan-sas-plus I }
    shows (\mp@subsup{\varphi}{S}{}\PsiG)\subseteq\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan ( }\mp@subsup{\varphi}{S}{}\PsiI)(\mp@subsup{\varphi}{P}{}\Psi\psi)
    proof -
let ?\Pi=\varphi\Psi
and ?G' = \varphi }\mp@subsup{\mp@code{S}}{}{\Psi}
show ?thesis
using assms
proof (induction \psi arbitrary: I)
case Nil
let ? I' = \varphi S \Psi I
have G}\mp@subsup{\subseteq}{m}{}
using Nil
by simp
moreover have ?G' }\mp@subsup{\subseteq}{m}{m}\mathrm{ ? I'
using state-to-strips-state-map-le-iff[OF Nil.prems(1, 4, 5)]
calculation..
ultimately show ?case
unfolding SAS-Plus-STRIPS.sas-plus-parallel-plan-to-strips-parallel-plan-def
sas-plus-parallel-plan-to-strips-parallel-plan-def
by simp
next
case (Cons ops \psi)
let ?vs = variables-of }
and ?ops = operators-of }
and ?J = execute-parallel-operator-sas-plus I ops
and ? }\pi=\mp@subsup{\varphi}{P}{}\Psi(ops\#\psi
let ? I' = \varphi < \Psi I
and ?J' = \varphi }\mp@subsup{\mp@code{S}}{}{\Psi}\Psi?
and ?ops' = [ [ }\mp@subsup{\varphi}{O}{}\Psi\mathrm{ op.op }\leftarrowops
{
fix op va
assume op fet ops and (v,a)\in set (effect-of op)
moreover have op \in set ?ops
using Cons.prems(6) calculation(1)
by simp
moreover have is-valid-operator-sas-plus \Psi op
using is-valid-problem-sas-plus-then(2) Cons.prems(1) calculation(3)
unfolding is-valid-operator-sas-plus-def
by auto
ultimately have v\in set ((\Psi)}\mp@subsup{)}{\mathcal{V}+}{}
and }a\in\mp@subsup{\mathcal{R}}{+}{}\Psi
using is-valid-operator-sas-plus-then (3,4)
by fastforce+
} note n\mp@subsup{b}{1}{}= this
show ?case
proof (cases are-all-operators-applicable-in I ops
\wedge ~ a r e - a l l - o p e r a t o r - e f f e c t s - c o n s i s t e n t ~ o p s )
case True

```

\section*{\{}
have \(\left(\varphi_{P} \Psi(\right.\) ops \(\left.\# \psi)\right)=\) ? ops \({ }^{\prime} \#\left(\varphi_{P} \Psi \psi\right)\)
unfolding sas-plus-parallel-plan-to-strips-parallel-plan-def SAS-Plus-STRIPS.sas-plus-parallel-plan-to-strips-parallel-plan-def sasp-op-to-strips-def SAS-Plus-STRIPS.sasp-op-to-strips-def by \(\operatorname{simp}\)
moreover have \(\forall o p \in\) set ops. op \(\in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)
using Cons.prems(6)
by \(\operatorname{simp}\)
moreover have STRIPS-Semantics.are-all-operators-applicable ? \(I^{\prime}\) ?ops \({ }^{\prime}\)
and STRIPS-Semantics.are-all-operator-effects-consistent ?ops'
using strips-equivalent-to-sas-plus-i-a-IV[OF Cons.prems(1) - True]
calculation
by blast+
ultimately have execute-parallel-plan ? \(I^{\prime} ? \pi\)
\(=\) execute-parallel-plan (execute-parallel-operator ? \(I^{\prime}\) ?ops') \(\left(\varphi_{P} \Psi \psi\right)\)
by fastforce
\}
- NOTE Instantiate the IH on the next state of the SAS+ execution execute-parallel-operator-sas-plus I ops.

\section*{moreover}
\{
have dom \(I \subseteq\) set (sas-plus-problem.variables-of \(\Psi\) )
using Cons.prems(2)
by blast
moreover have \(\forall o p \in\) set ops. \(\forall(v, a) \in\) set (effect-of op).
\(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
using \(n b_{1}(1)\)
by blast
ultimately have \(d o m ? J \subseteq \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
using sas-plus-equivalent-to-strips-i-a-IX[of I set?vs]
by \(\operatorname{simp}\)
\} note \(n b_{2}=\) this
moreover \{
have dom \(I \subseteq\) set (sas-plus-problem.variables-of \(\Psi\) )
using Cons.prems(2)
by blast
moreover have set (sas-plus-problem.variables-of \(\Psi\) )
\(\subseteq\) dom (range-of \(\Psi\) )
using is-valid-problem-sas-plus-dom-sas-plus-problem-range-of assms(1) by auto
moreover \{
fix \(v\)
assume \(v \in \operatorname{dom} I\)
moreover have \(v \in \operatorname{set}\left((\Psi)_{\mathcal{V}_{+}}\right)\)
using Cons.prems(2) calculation
by blast
```

            ultimately have the (Iv)\in set (the (range-of \Psi v))
            using Cons.prems(3)
            using set-the-range-of-is-range-of-sas-plus-if[OF assms(1)]
            by blast
        }
    moreover have }\forallop\inset ops. \forall(v,a)\inset (effect-of op)
        v \in \text { set (sas-plus-problem.variables-of } \Psi \text { ) } \wedge a \in \text { set (the (range-of } \Psi
    v))
using set-the-range-of-is-range-of-sas-plus-if[OF assms(1)] n\mp@subsup{b}{1}{}(1)
nb
by force
moreover have n\mp@subsup{b}{3}{}:\forallv\indom?J. the (?J v)\in set (the (range-of \Psi
v))
using sas-plus-equivalent-to-strips-i-a-X[of I set ?vs range-of \Psi ops]
calculation
by fast
moreover {
fix }
assume v\indom ?J
moreover have v\in set ((\Psi)}\mp@subsup{\mathcal{V}}{+}{}
using n\mp@subsup{b}{2}{}}\mathrm{ calculation
by blast
moreover have set (the (range-of \Psi v)) = \mathcal{R}}
using set-the-range-of-is-range-of-sas-plus-if[OF assms(1)]
calculation(2)
by presburger

```

```

            using nb 3
            by blast
        }
        ultimately have }\forallv\indom?J. the (?J v)\in\mathcal{R}+\Psi \Psi
            by fast
    }
    moreover have }\forallops\inset \psi.\forallop\inset ops.op\in set ?op
        using Cons.prems(6)
        by auto
    moreover have G}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan-sas-plus ?J }
        using Cons.prems(7) True
        by simp
    ultimately have ( }\mp@subsup{\varphi}{S}{}\PsiG)\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan ?J' ( }\mp@subsup{\varphi}{P}{}\Psi\psi
        using Cons.IH[of ?J, OF Cons.prems(1) - Cons.prems(4, 5)]
        by fastforce
    }
    moreover have execute-parallel-operator ? 'I' ?ops' = ? J'
    using assms(1) strips-equivalent-to-sas-plus-i-a-III[OF assms(1)] Cons.prems(6)
by auto
ultimately show ?thesis
by argo
next

```
```

        case False
        then have nb: G \subseteq}\mp@subsup{\}{m}{}
            using Cons.prems(7)
            by force
            moreover {
            have ? }\pi=?\mp@code{ops' # ( }\mp@subsup{\varphi}{P}{}\Psi\psi
                unfolding sas-plus-parallel-plan-to-strips-parallel-plan-def
                    SAS-Plus-STRIPS.sas-plus-parallel-plan-to-strips-parallel-plan-def
                    sasp-op-to-strips-def
                    SAS-Plus-STRIPS.sasp-op-to-strips-def Let-def
            by auto
            moreover have set ?ops' }\subseteq\mathrm{ set (strips-problem.operators-of ?П)
            using strips-equivalent-to-sas-plus-i-a-II (1)[OF assms(1)] Cons.prems(6)
            by auto
        moreover have }\neg(STRIPS-Semantics.are-all-operators-applicable ?I' ?ops'
            ^STRIPS-Semantics.are-all-operator-effects-consistent ?ops')
            using strips-equivalent-to-sas-plus-i-a-V[OF assms(1)- False] Cons.prems(6)
                by force
            ultimately have execute-parallel-plan ? I' ? | = ? I'
                by auto
            }
            moreover have ? }\mp@subsup{G}{}{\prime}\mp@subsup{\subseteq}{m}{}\mp@subsup{}{}{\prime}\mp@subsup{I}{}{\prime
            using state-to-strips-state-map-le-iff[OF Cons.prems(1, 4, 5)] nb
            by blast
            ultimately show ?thesis
                by presburger
            qed
        qed
    qed
lemma strips-equivalent-to-sas-plus-i:
assumes is-valid-problem-sas-plus \Psi
and is-parallel-solution-for-problem \Psi \psi
shows (strips-problem.goal-of (\varphi\Psi)) \subseteqm execute-parallel-plan
(strips-problem.initial-of ( }\varphi\Psi))(\mp@subsup{\varphi}{P}{}\Psi\psi
proof -
let ?vs = variables-of }
and ?ops = operators-of }
and ?I = initial-of }
and ?G= goal-of \Psi
let ?\Pi=\varphi\Psi
let ?'I' = strips-problem.initial-of ?\Pi
and ?G' = strips-problem.goal-of ?\Pi
have dom ?I \subseteq set ?vs
using is-valid-problem-sas-plus-then(3) assms(1)
by auto
moreover have }\forallv\indom ?I. the (?I v)\in\mathcal{R

```
using is-valid-problem-sas-plus-then(4) assms(1) calculation by auto
moreover have dom ? \(G \subseteq\) set \(\left((\Psi)_{\mathcal{V}_{+}}\right)\)
using is-valid-problem-sas-plus-then(5) assms(1)
by auto
moreover have \(\forall v \in\) dom? \(G\). the \((? G v) \in \mathcal{R}_{+} \Psi v\)
using is-valid-problem-sas-plus-then(6) assms(1)
by auto
moreover have \(\forall o p s \in\) set \(\psi . \forall o p \in\) set ops. op \(\in\) set ?ops
using is-parallel-solution-for-problem-plan-operator-set[OF assms(2)]
by fastforce
moreover have ? \(G \subseteq_{m}\) execute-parallel-plan-sas-plus ?I \(\psi\)
using assms(2)
unfolding is-parallel-solution-for-problem-def
by \(\operatorname{simp}\)
ultimately show ?thesis
using strips-equivalent-to-sas-plus-i-a \([O F \operatorname{assms}(1)\), of ?I ? \(G \psi]\)
unfolding sas-plus-problem-to-strips-problem-def
SAS-Plus-STRIPS.sas-plus-problem-to-strips-problem-def
state-to-strips-state-def
SAS-Plus-STRIPS.state-to-strips-state-def
by force
qed
lemma strips-equivalent-to-sas-plus-ii:
assumes is-valid-problem-sas-plus \(\Psi\)
and is-parallel-solution-for-problem \(\Psi \psi\)
shows list-all (list-all ( (lop. ListMem op (strips-problem.operators-of \((\varphi \Psi))\) ))
\(\left(\varphi_{P} \Psi \psi\right)\)
proof -
let ?ops \(=\) operators-of \(\Psi\)
let \(? \Pi=\varphi \Psi\)
let ?ops \({ }^{\prime}=\) strips-problem.operators-of ? \(\Pi\)
and \(? \pi=\varphi_{P} \Psi \psi\)
have is-valid-problem-strips ? \(\Pi\)
using is-valid-problem-sas-plus-then-strips-transformation-too[OF assms(1)]
by \(\operatorname{simp}\)
have \(n b_{1}: \forall o p \in\) set ?ops. \(\left(\exists o p^{\prime} \in\right.\) set ?ops'. \(\left.o p^{\prime}=\left(\varphi_{O} \Psi o p\right)\right)\)
unfolding sas-plus-problem-to-strips-problem-def SAS-Plus-STRIPS.sas-plus-problem-to-strips-problem-def Let-def sasp-op-to-strips-def
by force
\{
fix ops op \(o p^{\prime}\)
assume ops \(\in\) set \(\psi\) and op \(\in\) set ops
moreover have op \(\in \operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)\)
using is-parallel-solution-for-problem-plan-operator-set[OF assms(2)]
```

            calculation
        by blast
    moreover obtain }o\mp@subsup{p}{}{\prime}\mathrm{ where op' & set ?ops' and op' = ( }\mp@subsup{\varphi}{O}{\prime}\Psiop
        using n\mp@subsup{b}{1}{}\mathrm{ calculation(3)}
        by auto
    ultimately have ( }\mp@subsup{\varphi}{O}{}\Psiop)\in\mathrm{ set ?ops'
        by blast
    }
thus ?thesis
unfolding list-all-iff ListMem-iff Let-def
sas-plus-problem-to-strips-problem-def
SAS-Plus-STRIPS.sas-plus-problem-to-strips-problem-def
sas-plus-parallel-plan-to-strips-parallel-plan-def
SAS-Plus-STRIPS.sas-plus-parallel-plan-to-strips-parallel-plan-def
sasp-op-to-strips-def
SAS-Plus-STRIPS.sasp-op-to-strips-def
Let-def
by auto
qed

```

The following lemma proves the complementary proposition to theorem ??. Namely, given a parallel solution \(\psi\) for a SAS+ problem, the transformation to a STRIPS plan \(\varphi_{P} \Psi \psi\) also is a solution to the corresponding STRIPS problem \(\Pi \equiv \varphi \Psi\). In this direction, we have to show that the execution of the transformed plan reaches the goal state \(G^{\prime} \equiv \Pi_{G}\) of the corresponding STRIPS problem, i.e.
\[
G^{\prime} \subseteq_{m} \text { execute-parallel-plan } I^{\prime} \pi
\]
and that all operators in the transformed plan \(\pi\) are operators of \(\Pi\).
```

theorem
strips-equivalent-to-sas-plus:
assumes is-valid-problem-sas-plus $\Psi$
and is-parallel-solution-for-problem $\Psi \psi$
shows STRIPS-Semantics.is-parallel-solution-for-problem $(\varphi \Psi)\left(\varphi_{P} \Psi \psi\right)$
proof -
let $? \Pi=\varphi \Psi$
let $?^{\prime} I^{\prime}=$ strips-problem.initial-of ? $\Pi$
and $? G^{\prime}=$ strips-problem.goal-of ? $\Pi$
and ?ops' $=$ strips-problem.operators-of ? $\Pi$
and $? \pi=\varphi_{P} \Psi \psi$
show ?thesis
unfolding STRIPS-Semantics.is-parallel-solution-for-problem-def
proof (rule conjI)
show ? $G^{\prime} \subseteq_{m}$ execute-parallel-plan ? $I^{\prime} ? \pi$
using strips-equivalent-to-sas-plus-i[OF assms]
by $\operatorname{simp}$
next
show list-all (list-all ( $\lambda$ op. ListMem op ?ops')) ? $\pi$

```
```

            using strips-equivalent-to-sas-plus-ii[OF assms].
    qed
    qed
lemma embedded-serial-sas-plus-plan-operator-structure:
assumes ops \in set (embed \psi)
obtains op
where op f set \psi
and [\mp@subsup{\varphi}{O}{}\Psiop.op}\leftarrowops]=[\mp@subsup{\varphi}{O}{}\Psiop
proof -
let ? }\mp@subsup{\psi}{}{\prime}=\mathrm{ embed }
{
have ? }\mp@subsup{\psi}{}{\prime}=[[op].op\leftarrow\psi
by (induction \psi; force)
moreover obtain op where ops=[op] and op\in set \psi
using assms calculation
by fastforce
ultimately have }\existsop\inset \psi.[\mp@subsup{\varphi}{O}{}\Psiop.op\leftarrowops]=[[\mp@subsup{\varphi}{O}{\prime}\Psiop
by auto
}
thus ?thesis
using that
by meson
qed
private lemma serial-sas-plus-equivalent-to-serial-strips-i:
assumes ops }\in\operatorname{set}(\mp@subsup{\varphi}{P}{}\Psi(\mathrm{ embed }\psi)
obtains op where op set \psi and ops = [ [ O \Psi op]
proof -
let ? }\mp@subsup{\psi}{}{\prime}=\mathrm{ embed }
{
have set ( }\mp@subsup{\varphi}{P}{}\Psi(\mathrm{ embed }\psi))={[\mp@subsup{\varphi}{O}{}\Psi\mathrm{ op. op }\leftarrowops]| ops.ops\in set ? \psi ' }
unfolding sas-plus-parallel-plan-to-strips-parallel-plan-def
SAS-Plus-STRIPS.sas-plus-parallel-plan-to-strips-parallel-plan-def
sasp-op-to-strips-def set-map
using setcompr-eq-image
by blast

```

```

            using assms(1) calculation
            by blast
        moreover obtain op where op \in set \psi and ops = [ [ }\mp@subsup{\varphi}{O}{}
            using embedded-serial-sas-plus-plan-operator-structure calculation(2, 3)
            by blast
        ultimately have }\existsop\in\mathrm{ set }\psi\mathrm{ .ops = [ [ 
            by meson
        }
        thus ?thesis
    ```
```

    using that..
    ```
qed
private lemma serial-sas-plus-equivalent-to-serial-strips-ii \([\) simp \(]\) :
    concat \(\left(\varphi_{P} \Psi(\right.\) embed \(\left.\psi)\right)=\left[\varphi_{O} \Psi\right.\) op. op \(\left.\leftarrow \psi\right]\)
proof -
    let \(? \psi^{\prime}=\) List-Supplement.embed \(\psi\)
    have concat \(\left(\varphi_{P} \Psi ? \psi^{\prime}\right)=\operatorname{map}\left(\right.\) ( op.\(\varphi_{O} \Psi\) op \()\left(\right.\) concat ? \(\left.\psi^{\prime}\right)\)
            unfolding sas-plus-parallel-plan-to-strips-parallel-plan-def
                SAS-Plus-STRIPS.sas-plus-parallel-plan-to-strips-parallel-plan-def
                sasp-op-to-strips-def
                SAS-Plus-STRIPS.sasp-op-to-strips-def Let-def
                map-concat
        by blast
    also have \(\ldots=\operatorname{map}\left(\lambda o p . \varphi_{O} \Psi o p\right) \psi\)
        unfolding concat-is-inverse-of-embed \([\) of \(\psi]\)..
    finally show concat \(\left(\varphi_{P} \Psi(\right.\) embed \(\left.\psi)\right)=\left[\varphi_{O} \Psi\right.\) op. op \(\left.\leftarrow \psi\right]\).
qed

Having established the equivalence of parallel STRIPS and SAS+, we can now show the equivalence in the serial case. The proof combines the embedding theorem for serial SAS+ solutions (??), the parallel plan equivalence theorem ??, and the flattening theorem for parallel STRIPS plans (??). More precisely, given a serial SAS+ solution \(\psi\) for a SAS+ problem \(\Psi\), the embedding theorem confirms that the embedded plan List-Supplement.embed \(\psi\) is an equivalent parallel solution to \(\Psi\). By parallel plan equivalence, \(\pi \equiv\) \(\varphi_{P} \Psi\) List-Supplement.embed \(\psi\) is a parallel solution for the corresponding STRIPS problem \(\varphi \Psi\). Moreover, since List-Supplement.embed \(\psi\) is a plan consisting of singleton parallel operators, the same is true for \(\pi\). Hence, the flattening lemma applies and concat \(\pi\) is a serial solution for \(\varphi \Psi\). Since concat moreover can be shown to be the inverse of List-Supplement.embed, the term
\[
\text { concat } \pi=\operatorname{concat}\left(\varphi_{P} \Psi(\text { embed } \psi)\right)
\]
can be reduced to the intuitive form
\[
\pi=\left[\varphi_{O} \Psi \text { op. op } \leftarrow \psi\right]
\]
which concludes the proof.
```

theorem
serial-sas-plus-equivalent-to-serial-strips:
assumes is-valid-problem-sas-plus \Psi
and SAS-Plus-Semantics.is-serial-solution-for-problem \Psi \psi
shows STRIPS-Semantics.is-serial-solution-for-problem ( }\varphi\Psi)[\mp@subsup{\varphi}{O}{}\Psi op.op
\psi]
proof -
let ? }\mp@subsup{\psi}{}{\prime}=\mathrm{ embed }

```
and \(? \Pi=\varphi \Psi\)
let \(? \pi^{\prime}=\varphi_{P} \Psi ? \psi^{\prime}\)
let \(? \pi=\) concat \(? \pi^{\prime}\)
\{
have SAS-Plus-Semantics.is-parallel-solution-for-problem \(\Psi\) ? \(\psi^{\prime}\)
using execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus[OF assms]
by simp
hence STRIPS-Semantics.is-parallel-solution-for-problem? \(\Pi_{?} \pi^{\prime}\)
using strips-equivalent-to-sas-plus[OF assms(1)]
by \(\operatorname{simp}\)
\}
moreover have \(? \pi=\left[\varphi_{O} \Psi\right.\) op. \(\left.o p \leftarrow \psi\right]\)
by \(\operatorname{simp}\)
moreover have is-valid-problem-strips ? \(\Pi\)
using is-valid-problem-sas-plus-then-strips-transformation-too[OF assms(1)].
moreover have \(\forall o p s \in\) set ? \(\pi^{\prime} . \exists o p \in \operatorname{set} \psi\). ops \(=\left[\begin{array}{lll}\varphi_{O} & \Psi o p\end{array}\right]\)
using serial-sas-plus-equivalent-to-serial-strips-i \([o f-\Psi \psi]\)
by metis
ultimately show ?thesis
using STRIPS-Semantics.flattening-lemma[of ? T\(]\)
by metis
qed
lemma embedded-serial-strips-plan-operator-structure:
assumes ops \({ }^{\prime} \in\) set (embed \(\pi\) )
obtains \(o p\)
where \(o p \in\) set \(\pi\) and \(\left[\varphi_{O}{ }^{-1} \Pi\right.\) op. op \(\left.\leftarrow o p s^{\dagger}\right]=\left[\varphi_{O}{ }^{-1} \Pi o p\right]\)
proof -
let \(? \pi^{\prime}=\) embed \(\pi\)
\{
have \(? \pi^{\prime}=[[o p] . o p \leftarrow \pi]\)
by (induction \(\pi\); force)
moreover obtain \(o p\) where \(o p s^{\prime}=[o p]\) and \(o p \in\) set \(\pi\)
using calculation assms
by fastforce
ultimately have \(\exists o p \in\) set \(\pi .\left[\varphi_{O}^{-1} \Pi o p . o p \leftarrow o s^{\prime}\right]=\left[\varphi_{O}^{-1} \Pi o p\right]\)
by auto
\}
thus ?thesis
using that
by meson
qed
private lemma serial-strips-equivalent-to-serial-sas-plus-i:
assumes ops \(\in \operatorname{set}\left(\varphi_{P}{ }^{-1} \Pi(\right.\) embed \(\left.\pi)\right)\)
obtains \(o p\) where \(o p \in\) set \(\pi\) and ops \(=\left[\varphi_{O}{ }^{-1} \Pi o p\right]\)
proof -
let \(? \pi^{\prime}=\) embed \(\pi\)
```

{
have set (\mp@subsup{\varphi}{P}{-1}\Pi(\mathrm{ embed }\pi))={[\mp@subsup{\varphi}{O}{-1}\Pi\mathrm{ op.op }\leftarrowops]|ops.ops\in set? ? '
}
unfolding strips-parallel-plan-to-sas-plus-parallel-plan-def
SAS-Plus-STRIPS.strips-parallel-plan-to-sas-plus-parallel-plan-def
strips-op-to-sasp-def set-map
using setcompr-eq-image
by blast
moreover obtain ops' where ops'}\in\mathrm{ set ? ? }\mp@subsup{\pi}{}{\prime}\mathrm{ and ops = [ [ OO-1 ח op. op }
ops]
using assms(1) calculation
by blast
moreover obtain op where op \inset \pi and ops = [ }\mp@subsup{\varphi}{O}{-1}\Piop
using embedded-serial-strips-plan-operator-structure calculation(2, 3)
by blast
ultimately have }\existsop\in\mathrm{ set }\pi\mathrm{ . ops = [ }\mp@subsup{\varphi}{O}{-1}\Piop
by meson
}
thus ?thesis
using that..
qed
private lemma serial-strips-equivalent-to-serial-sas-plus-ii[simp]:
concat ( }\mp@subsup{\varphi}{P}{-1}\Pi(\mathrm{ embed }\pi))=[\mp@subsup{\varphi}{O}{-1}\Pi\mathrm{ op.op }\leftarrow\pi
proof -
let ?\mp@subsup{\pi}{}{\prime}=\mathrm{ List-Supplement.embed }\pi

```

```

        unfolding strips-parallel-plan-to-sas-plus-parallel-plan-def
            SAS-Plus-STRIPS.strips-parallel-plan-to-sas-plus-parallel-plan-def
            strips-op-to-sasp-def
            SAS-Plus-STRIPS.strips-op-to-sasp-def Let-def
            map-concat
        by simp
    also have ... = map (\lambdaop. \varphi}\mp@subsup{O}{O}{-1}\Pi\mathrm{ op) }
        unfolding concat-is-inverse-of-embed [of \pi]..
    finally show concat ( }\mp@subsup{\varphi}{P}{-1}\Pi(\mathrm{ embed }\pi))=[\mp@subsup{\varphi}{O}{-1}\Pi\mathrm{ op.op }\leftarrow\pi]
    qed

```

Using the analogous lemmas for the opposite direction, we can show the counterpart to theorem ?? which shows that serial solutions to STRIPS solutions can be transformed to serial SAS+ solutions via composition of embedding, transformation and flattening.
```

theorem
serial-strips-equivalent-to-serial-sas-plus:
assumes is-valid-problem-sas-plus \Psi
and STRIPS-Semantics.is-serial-solution-for-problem ( }\varphi\Psi)
shows SAS-Plus-Semantics.is-serial-solution-for-problem \Psi [ [ O O
\pi
proof -

```
```

let ? }\mp@subsup{\pi}{}{\prime}=\mathrm{ embed }
and ?\Pi=\varphi\Psi
let ? }\mp@subsup{\psi}{}{\prime}=\mp@subsup{\varphi}{P}{-1}\Psi? ?\mp@subsup{\pi}{}{\prime
let ? }\psi=\mathrm{ concat ? }\mp@subsup{\psi}{}{\prime
{
have STRIPS-Semantics.is-parallel-solution-for-problem ?\Pi ? |'
using embedding-lemma[OF
is-valid-problem-sas-plus-then-strips-transformation-too[OF assms(1)]
assms(2)].
hence SAS-Plus-Semantics.is-parallel-solution-for-problem \Psi? ' '
using sas-plus-equivalent-to-strips[OF assms(1)]
by simp
}
moreover have ? }\psi=[\mp@subsup{\varphi}{O}{-1}\Psi op.op\leftarrow\pi
by simp
moreover have is-valid-problem-strips ?\Pi
using is-valid-problem-sas-plus-then-strips-transformation-too[OF assms(1)].
moreover have }\forallops\in set ? ' ''. \existsop \in set \pi. ops = [\varphi\mp@subsup{\varphi}{O}{-1}\Psi op
using serial-strips-equivalent-to-serial-sas-plus-i
by metis
ultimately show ?thesis
using flattening-lemma[OF assms(1)]
by metis
qed

```

\subsection*{6.2 Equivalence of SAS+ and STRIPS}
abbreviation bounded-plan-set
    where bounded-plan-set ops \(k \equiv\{\pi\). set \(\pi \subseteq\) set ops \(\wedge\) length \(\pi=k\}\)
definition bounded-solution-set-sas-plus'
:: ('variable, 'domain) sas-plus-problem
\(\Rightarrow\) nat
\(\Rightarrow\) ('variable, 'domain) sas-plus-plan set
where bounded-solution-set-sas-plus' \(\Psi k\)
\(\equiv\{\psi\). is-serial-solution-for-problem \(\Psi \psi \wedge\) length \(\psi=k\}\)
abbreviation bounded-solution-set-sas-plus
:: ('variable, 'domain) sas-plus-problem
\(\Rightarrow\) nat
\(\Rightarrow\) ('variable, 'domain) sas-plus-plan set
where bounded-solution-set-sas-plus \(\Psi N\)
\[
\equiv\left(\bigcup k \in\{0 . . N\} . \text { bounded-solution-set-sas-plus }{ }^{\prime} \Psi k\right)
\]
definition bounded-solution-set-strips \({ }^{\prime}\)
:: ('variable \(\times\) 'domain) strips-problem
\(\Rightarrow\) nat
\(\Rightarrow\) ('variable \(\times\) 'domain) strips-plan set
where bounded-solution-set-strips \({ }^{\prime} \Pi k\)
```

\equiv{\pi.STRIPS-Semantics.is-serial-solution-for-problem \Pi\pi\wedge length \pi=k}

```
abbreviation bounded-solution-set-strips
:: ('variable \(\times\) 'domain) strips-problem
\(\Rightarrow\) nat
\(\Rightarrow\) ('variable \(\times\) 'domain) strips-plan set
where bounded-solution-set-strips \(\Pi N \equiv\left(\bigcup k \in\{0 . . N\}\right.\). bounded-solution-set-strips \({ }^{\prime}\) \(\Pi k)\)
- Show that plan transformation for all SAS Plus solutions yields a STRIPS solution for the induced STRIPS problem with same length.
We first show injectiveness of plan transformation \(\lambda \psi\). [ \(\varphi_{O} \Psi\) op. op \(\left.\leftarrow \psi\right]\) on the set of plans \(P_{k} \equiv\) bounded-plan-set (operators-of \(\Psi\) ) \(k\) with length bound \(k\). The injectiveness of \(S o l_{k} \equiv\) bounded-solution-set-sas-plus \(\Psi k\)-the set of solutions with length bound \(k\)-then follows from the subset relation \(S o l_{k} \subseteq P_{k}\).
lemma sasp-op-to-strips-injective:
```

assumes $\left(\varphi_{O} \Psi o p_{1}\right)=\left(\varphi_{O} \Psi o p_{2}\right)$
shows $o p_{1}=o p_{2}$
proof -
let ${ }^{\circ} o p_{1}{ }^{\prime}=\varphi_{O} \Psi o p_{1}$
and $? o p_{2}{ }^{\prime}=\varphi_{O} \Psi o p_{2}$
\{
have strips-operator.precondition-of ?op $_{1}{ }^{\prime}=$ strips-operator.precondition-of
? op ${ }^{\prime}$
using assms
by argo
hence sas-plus-operator.precondition-of op ${ }_{1}=$ sas-plus-operator.precondition-of
$o p_{2}$
unfolding sasp-op-to-strips-def
SAS-Plus-STRIPS.sasp-op-to-strips-def
Let-def
by $\operatorname{simp}$
\}
moreover \{
have strips-operator.add-effects-of ?op ${ }_{1}{ }^{\prime}=$ strips-operator.add-effects-of ?op ${ }_{2}{ }^{\prime}$
using assms
unfolding sasp-op-to-strips-def Let-def
by argo
hence sas-plus-operator.effect-of op $p_{1}=$ sas-plus-operator.effect-of op $p_{2}$
unfolding sasp-op-to-strips-def Let-def
SAS-Plus-STRIPS.sasp-op-to-strips-def
by $\operatorname{simp}$
\}
ultimately show ?thesis
by $\operatorname{simp}$
qed

```
lemma sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-i-a: assumes is-valid-problem-sas-plus \(\Psi\)
```

shows inj-on ( }\lambda\psi.[\mp@subsup{\varphi}{O}{}\Psi\mathrm{ op.op }\leftarrow\psi])\mathrm{ (bounded-plan-set (sas-plus-problem.operators-of
\Psi)k)
proof -
let ?ops= sas-plus-problem.operators-of \Psi
and ? }\mp@subsup{\varphi}{P}{}=\lambda\psi.[\mp@subsup{\varphi}{O}{}\Psi\mathrm{ Iop. op }\leftarrow\psi
let ?P = bounded-plan-set ?ops
{
fix }\mp@subsup{\psi}{1}{}\mp@subsup{\psi}{2}{
assume \psi \psi -in: }\mp@subsup{\psi}{1}{}\in?PP
and \psi}\mp@subsup{\psi}{2}{}-in:\mp@subsup{\psi}{2}{}\in?P\mp@code{P
and }\mp@subsup{\varphi}{P}{}-of-\mp@subsup{\psi}{1}{}-is-\mp@subsup{\varphi}{P}{}-of-\mp@subsup{\psi}{2}{}:(?\mp@subsup{\varphi}{P}{}\quad\mp@subsup{\psi}{1}{})=(?\mp@subsup{\varphi}{P}{}\mp@subsup{\psi}{2}{}
hence }\mp@subsup{\psi}{1}{}=\mp@subsup{\psi}{2}{
proof (induction k arbitrary: \psi }\mp@subsup{\psi}{1}{}\mp@subsup{\psi}{2}{}\mathrm{ )
case 0
then have length }\mp@subsup{\psi}{1}{}=
and length }\mp@subsup{\psi}{2}{}=
using }\mp@subsup{\psi}{1}{}-in \mp@subsup{\psi}{2}{}-i
unfolding bounded-solution-set-sas-plus'-def
by blast+
then show ?case
by blast
next
case (Suc k)
moreover have length }\mp@subsup{\psi}{1}{}=\mathrm{ Suc k and length }\mp@subsup{\psi}{2}{}=\mathrm{ Suc k
using length-Suc-conv Suc(2, 3)
unfolding bounded-solution-set-sas-plus'-def
by blast+
moreover obtain op }\mp@subsup{\psi}{1}{}\mp@subsup{}{}{\prime}\mathrm{ where }\mp@subsup{\psi}{1}{}=o\mp@subsup{p}{1}{}\#\mp@subsup{\psi}{1}{}\mp@subsup{}{}{\prime
and set (o\mp@subsup{p}{1}{}\#}\mp@subsup{\psi}{1}{\prime})\subseteq\mathrm{ set ?ops
and length }\mp@subsup{\psi}{1}{\prime}\mp@subsup{}{}{\prime}=
using calculation(5) Suc(2)
unfolding length-Suc-conv
by blast
moreover obtain op }\mp@subsup{|}{2}{}\mp@subsup{\psi}{2}{\prime}\mathrm{ ' where }\mp@subsup{\psi}{2}{}=o\mp@subsup{p}{2}{}\#\mp@subsup{\psi}{2}{\prime
and set (op \# \# \psi + ') \subseteq set ?ops
and length }\mp@subsup{\psi}{2}{\prime}\mp@subsup{}{}{\prime}=
using calculation(6) Suc(3)
unfolding length-Suc-conv
by blast
moreover have set }\mp@subsup{\psi}{1}{\prime}\subseteq\mathrm{ set ?ops and set }\mp@subsup{\psi}{2}{\prime}\subseteq\mathrm{ set ?ops
using calculation(8, 11)
by auto+
moreover have }\mp@subsup{\psi}{1}{\prime}\mp@subsup{}{}{\prime}\in?P\mp@code{F}\mathrm{ and }\mp@subsup{\psi}{2}{\prime}\mp@subsup{}{}{\prime}\in?P\mp@code{R
using calculation(9, 12, 13, 14)
by fast+
moreover have ? }\mp@subsup{\varphi}{P}{}\mp@subsup{\psi}{1}{}\mp@subsup{}{}{\prime}=? ?\mp@subsup{\varphi}{P}{}\mp@subsup{\psi}{2}{}\mp@subsup{}{}{\prime
using Suc.prems(3) calculation(7, 10)
by fastforce

```
```

            moreover have }\mp@subsup{\psi}{1}{\prime}=\mp@subsup{\psi}{2}{}\mp@subsup{}{}{\prime
            using Suc.IH[of }\mp@subsup{\psi}{1}{\prime}\mp@subsup{}{}{\prime}\mp@subsup{\psi}{2}{\prime}\mp@subsup{}{}{\prime},\mathrm{ OF calculation(15, 16, 17)]
            by simp
            moreover have ? }\mp@subsup{\varphi}{P}{}\mp@subsup{\psi}{1}{}=(\mp@subsup{\varphi}{O}{}\Psio\mp@subsup{p}{1}{})#?\mp@subsup{\varphi}{P}{}\mp@subsup{\psi}{1}{}\mp@subsup{}{}{\prime
                and ? }\mp@subsup{\varphi}{P}{}\quad\mp@subsup{\psi}{2}{}=(\mp@subsup{\varphi}{O}{}\Psi|o\mp@subsup{p}{2}{})#?\mp@subsup{\varphi}{P}{}\mp@subsup{\psi}{2}{\prime
                    using Suc.prems(3) calculation(7, 10)
                    by fastforce+
            moreover have ( }\mp@subsup{\varphi}{O}{}\Psio\mp@subsup{p}{1}{})=(\mp@subsup{\varphi}{O}{}\Psio\mp@subsup{p}{2}{}
                    using Suc.prems(3) calculation(17, 19, 20)
                    by simp
            moreover have op ( = op 2
                    using sasp-op-to-strips-injective[OF calculation(21)].
            ultimately show ?case
                    by argo
    qed
    }
    thus ?thesis
        unfolding inj-on-def
        by blast
    qed
    private corollary sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-i-b:
assumes is-valid-problem-sas-plus \Psi
shows inj-on ( }\lambda\psi.[\mp@subsup{\varphi}{O}{}\Psi\mathrm{ op. op }\leftarrow\psi])(bounded-solution-set-sas-plus' \Psi k
proof -
let ?ops = sas-plus-problem.operators-of \Psi
and ? }\mp@subsup{\varphi}{P}{}=\lambda\psi.[\mp@subsup{\varphi}{O}{}\Psi\mathrm{ I op. op }\leftarrow\psi
{
fix }
assume \psi \in bounded-solution-set-sas-plus' \Psi k
then have set \psi\subseteq set ?ops
and length }\psi=
unfolding bounded-solution-set-sas-plus'-def is-serial-solution-for-problem-def
Let-def
list-all-iff ListMem-iff
by fast+
hence }\psi\in\mathrm{ bounded-plan-set ?ops k
by blast
}
hence bounded-solution-set-sas-plus' }\Psik\subseteq\mathrm{ bounded-plan-set ?ops k
by blast
moreover have inj-on ? }\mp@subsup{\varphi}{P}{}\mathrm{ (bounded-plan-set ?ops k)
using sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-i-a[OF
assms(1)].
ultimately show ?thesis
using inj-on-subset[ of ? }\mp@subsup{\varphi}{P}{}\mathrm{ bounded-plan-set ?ops k bounded-solution-set-sas-plus'
\Psik]
by fast
qed

```
- Show that mapping plan transformation \(\lambda \psi\). \(\left[\varphi_{O} \Psi o p . o p \leftarrow \psi\right]\) over the solution set for a given SAS+ problem yields the solution set for the induced STRIPS problem.
private lemma sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-i-c: assumes is-valid-problem-sas-plus \(\Psi\)
shows \(\left(\lambda \psi .\left[\varphi_{O} \Psi\right.\right.\) op. op \(\left.\left.\leftarrow \psi\right]\right)\) ' (bounded-solution-set-sas-plus' \(\left.\Psi k\right)\)
\(=\) bounded-solution-set-strips \({ }^{\prime}(\varphi \Psi) k\)
proof -
let ? \(\Pi=\varphi \Psi\)
and \(? \varphi_{P}=\lambda \psi \cdot\left[\varphi_{O} \Psi\right.\) op. op \(\left.\leftarrow \psi\right]\)
let ? Sol \(_{k}=\) bounded-solution-set-sas-plus \({ }^{\prime} \Psi k\)
and \({ }^{2}\) Sol \(_{k}{ }^{\prime}=\) bounded-solution-set-strips \({ }^{\prime}\) ? \({ }^{\prime} k\)
\{
assume \({ }^{?} \varphi_{P}{ }^{\prime} ?{ }^{?} S^{\prime} l_{k} \neq ?\) Sol \(_{k}{ }^{\prime}\)

\(\mid(B) \exists \pi \in ?^{\prime} S_{o l}{ }_{k}{ }^{\prime} . \pi \notin ?^{\prime} \varphi_{P}{ }^{\prime}{ }^{?}\) ?Sol \(_{k}\)
by blast
hence False
proof (cases)
case \(A\)
moreover obtain \(\pi\) where \(\pi \in ? \varphi_{P}{ }^{\prime} ?\) Sol \(_{k}\) and \(\pi \notin ?\) Sol \(_{k}{ }^{\prime}\)
using calculation
by blast
moreover obtain \(\psi\) where length \(\psi=k\)
and SAS-Plus-Semantics.is-serial-solution-for-problem \(\Psi \psi\)
and \(\pi=? \varphi_{P} \psi\)
using calculation(2)
unfolding bounded-solution-set-sas-plus'-def
by blast
moreover have length \(\pi=k\) and STRIPS-Semantics.is-serial-solution-for-problem
?П \(\pi\)
subgoal
using calculation \((4,6)\) by auto
subgoal
using serial-sas-plus-equivalent-to-serial-strips
\(\operatorname{assms}(1)\) calculation(5) calculation(6)
by blast
done
moreover have \(\pi \in\) ? Sol \(_{k}{ }^{\prime}\)
unfolding bounded-solution-set-strips'-def
using calculation (7, 8)
by \(\operatorname{simp}\)
ultimately show ?thesis
by fast
next
case \(B\)
moreover obtain \(\pi\) where \(\pi \in ?\) \(S o l_{k}{ }^{\prime}\) and \(\pi \notin ? \varphi_{P}{ }^{\text {' } ? S o l_{k}}\)
using calculation
by blast
moreover have STRIPS-Semantics.is-serial-solution-for-problem ? \(\Pi \pi\)
and length \(\pi=k\)
using calculation(2)
unfolding bounded-solution-set-strips'-def
by \(\operatorname{simp}+\)
- Construct the counter example \(\psi \equiv\left[\varphi_{O}^{-1}\right.\) ? \(\Pi\) op. op \(\left.\leftarrow \pi\right]\) and show

moreover have length \(\left[\varphi_{O}{ }^{-1} \Psi\right.\) op. op \(\left.\leftarrow \pi\right]=k\)
and SAS-Plus-Semantics.is-serial-solution-for-problem \(\Psi\left[\varphi_{O}{ }^{-1} \Psi\right.\) op. op \(\leftarrow \pi]\)
subgoal
using calculation(5)
by simp
subgoal
using serial-strips-equivalent-to-serial-sas-plus[OF assms(1)] calculation(4)
by \(\operatorname{simp}\)
done
moreover have \(\left[\varphi_{O}{ }^{-1} \Psi\right.\) op. op \(\left.\leftarrow \pi\right] \in\) ? \(S_{o l} l_{k}\)
unfolding bounded-solution-set-sas-plus'-def
using calculation ( 6,7 )
by blast
moreover \{
have \(\forall o p \in \operatorname{set} \pi\). op \(\in \operatorname{set}\left((? \Pi)_{\mathcal{O}}\right)\)
using calculation(4)
unfolding STRIPS-Semantics.is-serial-solution-for-problem-def list-all-iff
ListMem-iff
by \(\operatorname{simp}\)
hence ? \(\varphi_{P}\left[\varphi_{O}{ }^{-1} \Psi\right.\) op. op \(\left.\leftarrow \pi\right]=\pi\)
proof (induction \(\pi\) )
case (Cons op \(\pi\) )
moreover have ? \(\varphi_{P}\left[\varphi_{O}{ }^{-1} \Psi\right.\) op. op \(\left.\leftarrow o p \# \pi\right]\)
\(=\left(\varphi_{O} \Psi\left(\varphi_{O}{ }^{-1} \Psi o p\right)\right) \# ? \varphi_{P}\left[\varphi_{O}{ }^{-1} \Psi\right.\) op. op \(\left.\leftarrow \pi\right]\)
by \(\operatorname{simp}\)
moreover have op \(\in \operatorname{set}\left((? \Pi)_{\mathcal{O}}\right)\)
using Cons.prems
by \(\operatorname{simp}\)
moreover have \(\left(\varphi_{O} \Psi\left(\varphi_{O}{ }^{-1} \Psi o p\right)\right)=o p\)
using strips-operator-inverse-is[OF assms(1) calculation(4)].
moreover have \(? \varphi_{P}\left[\varphi_{O}{ }^{-1} \Psi\right.\) op. op \(\left.\leftarrow \pi\right]=\pi\)
using Cons.IH Cons.prems
by auto
ultimately show ?case
by argo
qed \(\operatorname{simp}\)
```

            }
            moreover have }\pi\in?\mp@subsup{\varphi}{P}{\prime}\mp@subsup{}{}{\prime}?\mp@subsup{}{}{\prime}\mp@subsup{S}{ol}{k
                    using calculation(8,9)
                    by force
            ultimately show ?thesis
                    by blast
        qed
    }
    thus ?thesis
        by blast
    qed
    private lemma sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-i-d
assumes is-valid-problem-sas-plus \Psi
shows card (bounded-solution-set-sas-plus' \Psi k) \leq card (bounded-solution-set-strips'
(\varphi\Psi)k)
proof -
let ?\Pi=\varphi\Psi
and ?. }\mp@subsup{\varphi}{P}{}=\lambda\psi.[\mp@subsup{\varphi}{O}{}\Psi\mathrm{ op. op }\leftarrow\psi
let ?Sol}\mp@subsup{}{k}{}=\mathrm{ bounded-solution-set-sas-plus' }\Psi
and ?Solk}\mp@subsup{}{}{\prime}=\mathrm{ bounded-solution-set-strips' ?П k
have card (?\mp@subsup{\varphi}{P}{\prime}}\mp@subsup{}{}{\prime}?\mp@subsup{SOl}{k}{})=\operatorname{card}(?\mp@subsup{\mathrm{ Sol}}{k}{}
using sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-i-b[OF
assms(1)]
card-image
by blast
moreover have ? }\mp@subsup{\varphi}{P}{}\mp@subsup{}{}{\prime}\mp@subsup{}{}{\prime}\mathrm{ ? Sol }\mp@subsup{|}{k}{}=?\mp@subsup{?}{}{Sol}\mp@subsup{|}{k}{}\mp@subsup{}{}{\prime
using sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-i-c[OF
assms(1)].
ultimately show ?thesis
by simp
qed

- The set of fixed length plans with operators in a given operator set is finite.
lemma bounded-plan-set-finite:
shows finite {\pi. set }\pi\subseteq\mathrm{ set ops }\wedge\mathrm{ length }\pi=k
proof (induction k)
case (Suc k)
let ?P}={\pi\mathrm{ . set }\pi\subseteq\mathrm{ set ops }\wedge length \pi=k
and ?P'}={\pi\mathrm{ . set }\pi\subseteq\mathrm{ set ops ^ length }\pi=\mathrm{ Suc k }
let ?P'\prime}=(\bigcupop\in set ops. (\bigcup\pi\in?P.{op\#\pi }))
{
have }\forallop\pi\mathrm{ . finite {op \#
by simp
then have }\forallop.finite ( \bigcup\pi\in?P. {op\# \# }
using finite-UN[of ?P] Suc
by blast
hence finite?P'/
using finite-UN[of set ops]

```
```

        by blast
    }
    moreover {
    {
        fix }
        assume }\pi\in?\mp@subsup{P}{}{\prime
        moreover have set \pi\subseteq set ops
            and length \pi= Suc k
            using calculation
            by simp+
            moreover obtain op \mp@subsup{\pi}{}{\prime}}\mathrm{ where }\pi=op#\mp@subsup{\pi}{}{\prime
            using calculation (3)
            unfolding length-Suc-conv
            by fast
        moreover have set }\mp@subsup{\pi}{}{\prime}\subseteq\mathrm{ set ops and op set ops
            using calculation(2, 4)
            by simp+
    moreover have length }\mp@subsup{\pi}{}{\prime}=
            using calculation(3, 4)
            by auto
        moreover have }\mp@subsup{\pi}{}{\prime}\in?
            using calculation(5, 7)
            by blast
        ultimately have }\pi\in?\mp@subsup{P}{}{\prime\prime
            by blast
    }
    hence ?P'\subseteq?P'
        by blast
    }
    ultimately show ?case
    using rev-finite-subset[of ?P"' ?P]
    by blast
    qed force

- The set of fixed length SAS+ solutions are subsets of the set of plans with fixed length and therefore also finite.
private lemma sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-ii-a:
assumes is-valid-problem-sas-plus $\Psi$
shows finite (bounded-solution-set-sas-plus ${ }^{\prime} \Psi k$ )
proof -
let ? Ops $=\operatorname{set}\left((\Psi)_{\mathcal{O}_{+}}\right)$
let $?$ Sol $_{k}=$ bounded-solution-set-sas-plus' $\Psi k$
and $? P_{k}=\{\pi$. set $\pi \subseteq$ ?Ops $\wedge$ length $\pi=k\}$
\{
fix $\psi$
assume $\psi \in$ ? Sol $k_{k}$
then have length $\psi=k$ and set $\psi \subseteq$ ? Ops
unfolding bounded-solution-set-sas-plus'-def
SAS-Plus-Semantics.is-serial-solution-for-problem-def Let-def list-all-iff List-

```
```

Mem-iff
by fastforce+
hence }\psi\in??\mp@subsup{P}{k}{
by blast
}
then have ?Sol}\mp@subsup{\mp@code{k}}{}{\subseteq}\mathrm{ ? ?P}\mp@subsup{P}{k}{
by force
thus ?thesis
using bounded-plan-set-finite rev-finite-subset[of ? P}\mp@subsup{P}{k}{}\mathrm{ ?Sol }\mp@subsup{l}{k}{}
by auto
qed

```
- The set of fixed length STRIPS solutions are subsets of the set of plans with fixed length and therefore also finite.
private lemma sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-ii-b:
    assumes is-valid-problem-sas-plus \(\Psi\)
    shows finite (bounded-solution-set-strips' \((\varphi \Psi) k\) )
proof -
    let \(? \Pi=\varphi \Psi\)
    let ?Ops \(=\operatorname{set}\left((? \Pi)_{\mathcal{O}}\right)\)
    let ?Sol \({ }_{k}=\) bounded-solution-set-strips' \(? ~\) П \(k\)
        and \(? P_{k}=\{\pi\). set \(\pi \subseteq\) ?Ops \(\wedge\) length \(\pi=k\}\)
    \{
        fix \(\pi\)
        assume \(\pi \in\) ? Sol \(_{k}\)
        then have length \(\pi=k\) and set \(\pi \subseteq\) ? Ops
            unfolding bounded-solution-set-strips'-def
                STRIPS-Semantics.is-serial-solution-for-problem-def Let-def list-all-iff List-
Mem-iff
            by fastforce+
        hence \(\pi \in\) ? \(P_{k}\)
            by blast
    \}
    then have \({ }^{?} S_{o l} l_{k} \subseteq ?^{\prime} P_{k}\)
        by force
    thus ?thesis
        using bounded-plan-set-finite rev-finite-subset \(\left[\right.\) of ? \(P_{k}\) ?Sol \(\left.{ }_{k}\right]\)
        unfolding state-to-strips-state-def
            SAS-Plus-STRIPS.state-to-strips-state-def operators-of-def
        by blast
qed

With the results on the equivalence of SAS+ and STRIPS solutions, we can now show that given problems in both formalisms, the solution sets have the same size. This is the property required by the definition of planning formalism equivalence presented earlier in theorem ?? (??) and thus end up with the desired equivalence result.
The proof uses the finiteness and disjunctiveness of the solution sets for either problem to be able to equivalently transform the set cardinality over
the union of sets of solutions with bounded lengths into a sum over the cardinality of the sets of solutions with bounded length. Moreover, since we know that for each SAS+ solution with a given length an equivalent STRIPS solution exists in the solution set of the transformed problem with the same length, both sets must have the same cardinality.
Hence the cardinality of the SAS+ solution set over all lengths up to a given upper bound \(N\) has the same size as the solution set of the corresponding STRIPS problem over all length up to a given upper bound \(N\).
```

theorem
assumes is-valid-problem-sas-plus \Psi
shows card (bounded-solution-set-sas-plus \Psi N)
= card (bounded-solution-set-strips ( }\varphi\Psi)N
proof -
let ?\Pi=\varphi\Psi
and ?R = {0..N}
- Due to the disjoint nature of the bounded solution sets for fixed plan length for
different lengths, we can sum the individual set cardinality to obtain the cardinality
of the overall SAS+ resp. STRIPS solution sets.
have finite-R: finite ?R
by simp
moreover {
have }\forallk\in?R. finite (bounded-solution-set-sas-plus' \Psi k
using sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-iv-a[OF
assms(1)]..
moreover have }\forallj\in?R.\forallk\in?R.j\not=
\longrightarrow bounded-solution-set-sas-plus' \Psi
\cap bounded-solution-set-sas-plus' \Psi k = \{ \}
unfolding bounded-solution-set-sas-plus'-def
by blast
ultimately have card (bounded-solution-set-sas-plus \Psi N)
=(\sumk\in?R.card (bounded-solution-set-sas-plus' \Psi k))
using card-UN-disjoint
by blast
}
moreover {
have }\forallk\in?R. finite (bounded-solution-set-strips' ?\Pi k
using sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-it-b[OF
assms(1)]..
moreover have }\forallj\in?R..\forallk\in?R.j\not=
\longrightarrow ~ b o u n d e d - s o l u t i o n - s e t - s t r i p s ' ~ ? \Pi ~ j ~
\capbounded-solution-set-strips' ?\Pi k={}
unfolding bounded-solution-set-strips'-def
by blast

```
            ultimately have card (bounded-solution-set-strips ? \(\Pi\) )
```

            =(\sumk\in?R.card (bounded-solution-set-strips' ?\Pi k))
            using card-UN-disjoint
            by blast
    }
    moreover {
    fix }
    have card (bounded-solution-set-sas-plus' }\Psik
        card ((\lambda\psi.[ [ 
            ' bounded-solution-set-sas-plus' \Psi k)
    using sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-i-b[OF
    assms]
card-image[symmetric]
by blast
hence card (bounded-solution-set-sas-plus' }\Psi k
= card (bounded-solution-set-strips' ?\Pi k)
using sas-plus-formalism-and-induced-strips-formalism-are-equally-expressive-i-c[OF
assms]
by presburger
}
ultimately show ?thesis
by presburger
qed
end
end
theory SAT-Plan-Base
imports List-Index.List-Index
Propositional-Proof-Systems.Formulas
STRIPS-Semantics
Map-Supplement List-Supplement
CNF-Semantics-Supplement CNF-Supplement
begin

- Hide constant and notation for $(\perp)$ to prevent warnings.
hide-const (open) Orderings.bot-class.bot
no-notation Orderings.bot-class.bot $(\perp)$
- Hide constant and notation for $\left(\left(-^{+}\right)\right)$to prevent warnings.
hide-const (open) Transitive-Closure.trancl
no-notation Transitive-Closure.trancl ((-+) [1000] 999)
- Hide constant and notation for $\left(\left(-^{+}\right)\right)$to prevent warnings.
hide-const (open) Relation.converse
no-notation Relation.converse ((--1) [1000] 999)

```

\section*{7 The Basic SATPlan Encoding}

We now move on to the formalization of the basic SATPlan encoding (see ??).
The two major results that we will obtain here are the soundness and completeness result outlined in ?? in ??.
Let in the following \(\Phi \equiv\) encode-to-sat \(\Pi t\) denote the SATPlan encoding for a STRIPS problem \(\Pi\) and makespan \(t\). Let \(k<t\) and \(I \equiv(\Pi)_{I}\) be the initial state of \(\Pi, G \equiv(\Pi)_{G}\) be its goal state, \(\mathcal{V} \equiv(\Pi)_{\mathcal{V}}\) its variable set, and \(\mathcal{O} \equiv(\Pi)_{\mathcal{O}}\) its operator set.

\subsection*{7.1 Encoding Function Definitions}

Since the SATPlan encoding uses propositional variables for both operators and state variables of the problem as well as time points, we define a datatype using separate constructors —State \(k n\) for state variables resp. Operator \(k n\) for operator activation-to facilitate case distinction. The natural number values store the time index resp. the indexes of the variable or operator within their lists in the problem representation.
```

datatype sat-plan-variable $=$
State nat nat
| Operator nat nat

```

A SATPlan formula is a regular propositional formula over SATPlan variables. We add a type synonym to improve readability.
type-synonym sat-plan-formula \(=\) sat-plan-variable formula
We now continue with the concrete definitions used in the implementation of the SATPlan encoding. State variables are encoded as literals over SATPlan variables using the State constructor of .
```

definition encode-state-variable
:: nat $\Rightarrow$ nat $\Rightarrow$ bool option $\Rightarrow$ sat-plan-variable formula
where encode-state-variable $t k v \equiv$ case $v$ of
Some True $\Rightarrow$ Atom (State $t k$ )
| Some False $\Rightarrow \neg($ Atom $($ State $t k))$

```

The initial state encoding (definition ??) is a conjunction of state variable encodings \(A \equiv\) encode-state-variable \(0 n b\) with \(n \equiv\) index vs \(v\) and \(b \equiv I\) \(v=\) Some True for all \(v \in \mathcal{V}\). As we can see below, the same function but substituting the initial state with the goal state and zero with the makespan \(t\) produces the goal state encoding (??). Note that both functions construct a conjunction of clauses \(A \vee \perp\) for which it is easy to show that we can normalize to conjunctive normal form (CNF).
definition encode-initial-state
:: 'variable strips-problem \(\Rightarrow\) sat-plan-variable formula ( \(\Phi_{I}-99\) )
where encode-initial-state \(\Pi\)
\(\equiv\) let \(I=\) initial-of \(\Pi\)
; vs \(=\) variables-of \(\Pi\)
in \(\bigwedge(\) map ( \(\lambda v\). encode-state-variable \(0(\) index vs \(v)(I v) \vee \perp)\)
(filter ( \(\lambda v\). I \(v \neq\) None) \(v s)\) )
definition encode-goal-state
:: 'variable strips-problem \(\Rightarrow\) nat \(\Rightarrow\) sat-plan-variable formula \(\left(\Phi_{G}-99\right)\)
where encode-goal-state \(\Pi t\)
\(\equiv\) let
vs \(=\) variables-of \(\Pi\)
; \(G=\) goal-of \(\Pi\)
in \(\bigwedge(\) map \((\lambda v\). encode-state-variable \(t(\) index vs \(v)(G v) \vee \perp)\)
(filter \((\lambda v . G v \neq\) None) \(v s))\)
Operator preconditions are encoded using activation-implies-precondition formulation as mentioned in ??: i.e. for each operator \(o p \in \mathcal{O}\) and \(p \in\) set (precondition-of op) we have to encode
```

Atom (Operator k (index ops op)) }->\mathrm{ Atom (State k (index vs v))

```

We use the equivalent disjunction in the formalization to simplify conversion to CNF.
```

definition encode-operator-precondition
:: 'variable strips-problem
$\Rightarrow$ nat
$\Rightarrow$ 'variable strips-operator
$\Rightarrow$ sat-plan-variable formula
where encode-operator-precondition $\Pi t o p \equiv$ let
$v s=$ variables-of $\Pi$
; ops $=$ operators-of $\Pi$
in $\bigwedge($ map $(\lambda v$.
$\neg($ Atom $($ Operator $t($ index ops op $))) \vee$ Atom $($ State $t($ index vs $v)))$
(precondition-of op))
definition encode-all-operator-preconditions
:: 'variable strips-problem
$\Rightarrow$ 'variable strips-operator list
$\Rightarrow$ nat
$\Rightarrow$ sat-plan-variable formula
where encode-all-operator-preconditions $\Pi$ ops $t \equiv$ let
$l=$ List.product $[0 . .<t]$ ops
in foldr $(\wedge)(\operatorname{map}(\lambda(t, o p)$. encode-operator-precondition $\Pi t o p) l)(\neg \perp)$

```

Analogously to the operator precondition, add and delete effects of operators have to be implied by operator activation. That being said, we have to encode both positive and negative effects and the effect must be active at the following time point: i.e.
```

Atom (Operator k m) -> Atom (State (Suc k) n)

```
for add effects respectively
```

Atom(Operator k m) -> \negAtom (State (Suc k) n)

```
for delete effects. We again encode the implications as their equivalent disjunctions in definition ??.
```

definition encode-operator-effect
:: 'variable strips-problem
$\Rightarrow$ nat
$\Rightarrow$ 'variable strips-operator
$\Rightarrow$ sat-plan-variable formula
where encode-operator-effect $\Pi$ t op
三let
vs $=$ variables-of $\Pi$
; ops $=$ operators-of $\Pi$
in $\bigwedge(\operatorname{map}(\lambda v$.
$\neg($ Atom $($ Operator $t($ index ops op $)))$
$\vee$ Atom (State (Suc t) (index vs v)))
(add-effects-of op)
@ map ( $\lambda v$.
$\neg($ Atom $($ Operator $t($ index ops op $)))$
$\vee \neg($ Atom $($ State $($ Suc $t)($ index vs $v))))$
(delete-effects-of op))

```
definition encode-all-operator-effects
    :: 'variable strips-problem
    \(\Rightarrow\) 'variable strips-operator list
    \(\Rightarrow\) nat
    \(\Rightarrow\) sat-plan-variable formula
    where encode-all-operator-effects \(\Pi\) ops \(t\)
    \(\equiv\) let \(l=\) List.product \([0 . .<t]\) ops
        in foldr \((\wedge)(\operatorname{map}(\lambda(t, o p)\). encode-operator-effect \(\Pi t\) op \() l)(\neg \perp)\)
definition encode-operators
```

:: 'variable strips-problem $\Rightarrow$ nat $\Rightarrow$ sat-plan-variable formula
where encode-operators $\Pi t$
$\equiv$ let ops $=$ operators-of $\Pi$
in encode-all-operator-preconditions $\Pi$ ops $t \wedge$ encode-all-operator-effects $\Pi$
ops $t$

```

Definitions ?? and ?? similarly encode the negative resp. positive transition frame axioms as disjunctions.
```

definition encode-negative-transition-frame-axiom
:: 'variable strips-problem
$\Rightarrow$ nat
$\Rightarrow$ 'variable
$\Rightarrow$ sat-plan-variable formula

```
```

where encode-negative-transition-frame-axiom $\Pi t v$
$\equiv$ let vs $=$ variables-of $\Pi$
; ops $=$ operators-of $\Pi$
; deleting-operators $=$ filter $(\lambda o p$. ListMem $v($ delete-effects-of op $))$ ops
in $\neg($ Atom $($ State $t($ index vs $v)))$
$\vee($ Atom (State (Suc $t)$ (index vs $v$ ))
$\vee \bigvee(\operatorname{map}(\lambda o p . A t o m($ Operator $t$ (index ops op))) deleting-operators) $)$
definition encode-positive-transition-frame-axiom
:: 'variable strips-problem
$\Rightarrow$ nat
$\Rightarrow$ 'variable
$\Rightarrow$ sat-plan-variable formula
where encode-positive-transition-frame-axiom $\Pi t v$
$\equiv$ let $v s=$ variables-of $\Pi$
; ops $=$ operators-of $\Pi$
adding-operators $=$ filter $($ ( $o$ op. ListMem $v($ add-effects-of op $))$ ops
in (Atom (State $t$ (index vs $v$ ))
$\vee(\neg($ Atom $($ State $($ Suc $t)($ index vs $v)))$
$\vee \bigvee(\operatorname{map}(\lambda o p$. Atom $($ Operator $t($ index ops op $))$ ) adding-operators $))$ )
definition encode-all-frame-axioms
:: 'variable strips-problem $\Rightarrow$ nat $\Rightarrow$ sat-plan-variable formula
where encode-all-frame-axioms $\Pi t$
$\equiv$ let $l=$ List.product $[0 . .<t]$ (variables-of $\Pi$ )
in $\bigwedge(\operatorname{map}(\lambda(k, v)$. encode-negative-transition-frame-axiom $\Pi k v) l$
@ map $(\lambda(k, v)$. encode-positive-transition-frame-axiom $\Pi k v) l$ )

```

Finally, the basic SATPlan encoding is the conjunction of the initial state, goal state, operator and frame axiom encoding for all time steps. The functions and \({ }^{6}\) take care of mapping the operator precondition, effect and frame axiom encoding over all possible combinations of time point and operators resp. time points, variables, and operators.
```

definition encode-problem (\$ - -99)
where encode-problem \Pit
\equiv encode-initial-state \Pi
^(encode-operators \Pi t
^(encode-all-frame-axioms \Pit
\wedge(encode-goal-state \Pi t)))

```

\subsection*{7.2 Decoding Function Definitions}

Decoding plans from a valuation \(\mathcal{A}\) of a SATPlan encoding entails extracting all activated operators for all time points except the last one. We implement this by mapping over all \(k<t\) and extracting activated operators-i.e. operators for which the model valuates the respective operator encoding at

\footnotetext{
\({ }^{6}\) Not shown.
}
time \(k\) to true -into a parallel operator (see definition ??).
```

definition decode-plan'
:: 'variable strips-problem
$\Rightarrow$ sat-plan-variable valuation
$\Rightarrow$ nat
$\Rightarrow$ 'variable strips-operator list
where decode-plan' ${ }^{\prime}$ A $i$
$\equiv$ let ops $=$ operators-of $\Pi$
; vs $=$ map ( ( op. Operator $i($ index ops op $))($ remdups ops $)$
in map ( $\lambda v$. case $v$ of Operator $-k \Rightarrow$ ops ! $k$ ) (filter $\mathcal{A} v s)$

```
— We decode maps over range \(0, \ldots, t-1\) because the last operator takes effect in \(t\) and must therefore have been applied in step \(t-\left(1::^{\prime} a\right)\).
definition decode-plan
:: 'variable strips-problem
\(\Rightarrow\) sat-plan-variable valuation
\(\Rightarrow\) nat
\(\Rightarrow\) 'variable strips-parallel-plan ( \(\left.\Phi^{-1}--99\right)\)
where decode-plan \(\Pi \mathcal{A} t \equiv \operatorname{map}(\) decode-plan' \(\Pi \mathcal{A})[0 . .<t]\)
Similarly to the operator decoding, we can decode a state at time \(k\) from a valuation of of the SATPlan encoding \(\mathcal{A}\) by constructing a map from list of assignments \((v, \mathcal{A}(\) State \(k(\) index \(v s v))\) ) for all \(v \in \mathcal{V}\).
definition decode-state-at
:: 'variable strips-problem
\(\Rightarrow\) sat-plan-variable valuation
\(\Rightarrow\) nat
\(\Rightarrow\) 'variable strips-state \(\left(\Phi_{S}{ }^{-1}--99\right)\)
where decode-state-at \(\Pi \mathcal{A} k\)
\(\equiv\) let
vs \(=\) variables-of \(\Pi\)
; state-encoding-to-assignment \(=\lambda v .(v, \mathcal{A}(\) State \(k(\) index vs \(v)))\)
in map-of (map state-encoding-to-assignment vs)
We continue by setting up the context for the proofs of soundness and completeness.
definition encode-transitions ::'variable strips-problem \(\Rightarrow\) nat \(\Rightarrow\) sat-plan-variable formula ( \(\Phi_{T}\) - 99) where
encode-transitions \(\Pi t\)
\[
\begin{aligned}
& \equiv \text { SAT-Plan-Base.encode-operators } \Pi t \wedge \\
& \text { SAT-Plan-Base.encode-all-frame-axioms } \Pi t
\end{aligned}
\]
- Immediately proof the sublocale proposition for strips in order to gain access to definitions and lemmas.

\footnotetext{
\({ }^{7}\) This is handled by function decode_plan' (not shown).
}
        encode-goal-state-def decode-plan-def decode-state-at-def
    by \(\operatorname{simp}+\)

\section*{context}
begin
lemma encode-state-variable-is-lit-plus-if:
assumes is-valid-problem-strips \(\Pi\) and \(v \in\) dom \(s\)
shows is-lit-plus (encode-state-variable \(k\) (index (strips-problem.variables-of \(\Pi\) )
\(v)(s v))\)
proof -
have \(s v \neq\) None
using is-valid-problem-strips-initial-of-dom assms(2)
by blast
then consider (s-of-v-is-some-true) s \(v=\) Some True
( (s-of-v-is-some-false) s \(v=\) Some False
by fastforce
thus ?thesis
unfolding encode-state-variable-def by (cases, simp + )
qed
lemma is-cnf-encode-initial-state:
assumes is-valid-problem-strips \(\Pi\)
shows is-cnf ( \(\Phi_{I} \Pi\) )
proof -
let ? \(I=(\Pi)_{I}\)
and ?vs \(=\) strips-problem.variables-of \(\Pi\)
let ?l \(=\operatorname{map}(\lambda v\). encode-state-variable \(0(\) index ?vs \(v)(\) ?I \(v) \vee \perp)\) (filter ( \(\lambda v\). ?I \(v \neq\) None) ?vs)
\{
fix \(C\)
assume \(c\)-in-set-l: \(C \in\) set ?l
have set ?l \(=(\lambda v\). encode-state-variable \(0(\) index ?vs \(v)(? I v) \vee \perp)\) ' set (filter ( \(\lambda v\). ?I \(v \neq\) None) ?vs)
using set-map[of \(\lambda v\). encode-state-variable 0 (index ?vs v) (?I v) \(\vee \perp\)
filter ( \(\lambda v\). ?I \(v \neq\) None) ?vs]
by blast
then have set ?l \(=(\lambda v\). encode-state-variable \(0(\) index ? vs \(v)(? I v) \vee \perp)\) '
\(\{v \in\) set ?vs. ?I \(v \neq\) None \(\}\)
using set-filter[of \(\lambda v\). ?I \(v \neq\) None ?vs]
```

    by argo
    then obtain v
    where c-is: C= encode-state-variable 0 (index ?vs v) (?I v)}\vee
    and v-in-set-vs: v\in set ?vs
    and I-of-v-is-not-None:?I v}\not=\mathrm{ None
    using c-in-set-l
    by auto
    {
    have v\indom ?I
        using I-of-v-is-not-None
        by blast
    moreover have is-lit-plus (encode-state-variable 0 (index ?vs v) (?I v))
        using encode-state-variable-is-lit-plus-if[OF - calculation] assms(1)
        by blast
    moreover have is-lit-plus }
        by simp
    ultimately have is-disj C
        using c-is
        by force
    }
    hence is-cnf C
    unfolding encode-state-variable-def
    using c-is
    by fastforce
    }
thus ?thesis
unfolding encode-initial-state-def SAT-Plan-Base.encode-initial-state-def Let-def
initial-of-def
using is-cnf-BigAnd[of ?l]
by (smt is-cnf-BigAnd)
qed
lemma encode-goal-state-is-cnf:
assumes is-valid-problem-strips \Pi
shows is-cnf (encode-goal-state \Pit)
proof -
let ?I = (\Pi) I
and ?G = (\Pi)}
and ?vs = strips-problem.variables-of \Pi
let ?l = map (\lambdav. encode-state-variable t (index ?vs v) (?G v)\vee\perp)
(filter (\lambdav. ?G v}\not=\mathrm{ None) ?vs)
{
fix C
assume C \in set ?l
moreover {
have set ?l = (\lambdav. encode-state-variable t (index ?vs v) (?G v)\vee \perp)
' set (filter (\lambdav.?G v\not= None) ?vs)

```
```

        unfolding set-map
        by blast
    then have set ?l = { encode-state-variable t (index ?vs v) (?G v)\vee \perp
        |v.v\in set ?vs ^?G v\not= None }
        by auto
    }
    moreover obtain v where C-is: C = encode-state-variable t (index ?vs v)
    (?Gv)\vee\perp
and}v\in\mathrm{ set ?vs
and G-of-v-is-not-None: ?G v}\not=\mathrm{ None
using calculation(1)
by auto
moreover {
have v\in dom ?G
using G-of-v-is-not-None
by blast
moreover have is-lit-plus (encode-state-variable t (index ?vs v) (?G v))
using assms(1) calculation
by (simp add: encode-state-variable-is-lit-plus-if)
moreover have is-lit-plus }
by simp
ultimately have is-disj C
unfolding C-is
by force
}
ultimately have is-cnf C
by simp
}
thus ?thesis
unfolding encode-goal-state-def SAT-Plan-Base.encode-goal-state-def Let-def
using is-cnf-BigAnd[of ?l]
by simp
qed
private lemma encode-operator-precondition-is-cnf:
is-cnf (encode-operator-precondition \Pi k op)
proof -
let ?vs = strips-problem.variables-of \Pi
and ?ops = strips-problem.operators-of \Pi
let ?l = map (\lambdav.\neg (Atom (Operator k (index ?ops op))) \vee Atom (State k (index
?vs v)))
(precondition-of op)
{
have set ?l = (\lambdav. ᄀ(Atom (Operator k (index ?ops op))) \vee Atom (State k
(index ?vs v)))
' set (precondition-of op)
using set-map
by force

```
then have set ?l \(=\{\neg(\) Atom \((\) Operator \(k(\) index ?ops op \())) \vee\) Atom (State \(k\) (index ?vs v)
\(\mid v . v \in \operatorname{set}\) (precondition-of op) \}
using setcompr-eq-image[of
\(\lambda v . \neg(\) Atom \((\) Operator \(k(\) index ?ops op \())) \vee\) Atom (State \(k\) (index ?vs \(v))\)
\(\lambda v . v \in \operatorname{set}\) (precondition-of op)]
by simp
\(\}\) note set-l-is \(=\) this
\{
fix \(C\)
assume \(C \in\) set ?l
then obtain \(v\)
where \(v \in\) set (precondition-of op)
and \(C=\neg(\) Atom \((\) Operator \(k\) (index ?ops op) \()) \vee\) Atom (State \(k\) (index ?vs
v))
using set-l-is
by blast
hence is-cnf \(C\)
by simp
\}
thus ?thesis
unfolding encode-operator-precondition-def
using is-cnf-BigAnd[of ?l]
by meson
qed
private lemma set-map-operator-precondition[simp]:
set (map ( \(\lambda(k, o p)\). encode-operator-precondition \(\Pi k\) op) (List.product \([0 . .<t]\) ops))
\(=\{\) encode-operator-precondition \(\Pi k o p \mid k o p .(k, o p) \in(\{0 . .<t\} \times\) set ops \()\}\) proof -
let \(? l^{\prime}=\) List.product \([0 . .<t]\) ops
let ?fs \(=\operatorname{map}(\lambda(k, o p)\). encode-operator-precondition \(\Pi k\) op \() ? l^{\prime}\)
have set-l'-is: set \(? l^{\prime}=\{0 . .<t\} \times\) set ops
by \(\operatorname{simp}\)
moreover \{
have set ?fs \(=(\lambda(k, o p)\). encode-operator-precondition \(\Pi k\) op \()\)
' \((\{0 . .<t\} \times\) set ops \()\)
using set-map set-l'-is
by \(\operatorname{simp}\)
also have \(\ldots=\{\) encode-operator-precondition \(\Pi k o p \mid k o p .(k, o p) \in\{0 . .<t\}\)
\(\times\) set ops \(\}\)
using setcompr-eq-image
by fast
finally have set ? \(f_{s}=\{\) encode-operator-precondition \(\Pi k o p\)
\(\mid k\) op. \((k, o p) \in(\{0 . .<t\} \times\) set ops \()\}\)
by blast
\}
thus ?thesis
```

    by blast
    qed
private lemma is-cnf-encode-all-operator-preconditions:
is-cnf (encode-all-operator-preconditions \Pi (strips-problem.operators-of \Pi) t)
proof -
let ?l' = List.product [0..<t] (strips-problem.operators-of \Pi)

```

```

    have }\forallf\in\mathrm{ set ?fs. is-cnff
        using encode-operator-precondition-is-cnf
        by fastforce
    thus ?thesis
        unfolding encode-all-operator-preconditions-def
        using is-cnf-foldr-and-if[of ?fs]
        by presburger
    qed
private lemma set-map-or[simp]:
set (map (\lambdav.Av\veeBv)vs)={Av\veeBv|v.v\in set vs }
proof -
let ?l = map (\lambdav. A v\vee Bv) vs
have set ?l = (\lambdav.Av\veeBv)' set vs
using set-map
by force
thus ?thesis
using setcompr-eq-image
by auto
qed
private lemma encode-operator-effects-is-cnf-i:
is-cnf ($map (\lambdav. ( ᄀ (Atom (Operator t (index (strips-problem.operators-of \Pi)
op))))
    \vee Atom (State (Suc t) (index (strips-problem.variables-of \Pi) v))) (add-effects-of
op)))
proof -
    let ?fs = map ( }\lambdav.\neg(\mathrm{ Atom (Operator t (index (strips-problem.operators-of П)
op)))
    \vee Atom (State (Suc t) (index (strips-problem.variables-of \Pi) v))) (add-effects-of
op)
    {
        fix C
        assume C set ?fs
        then obtain v
            where v}\mathrm{ set (add-effects-of op)
                        and C = ᄀ(Atom (Operator t(index (strips-problem.operators-of \Pi) op)))
                        \vee ~ A t o m ~ ( S t a t e ~ ( S u c ~ t ) ~ ( i n d e x ~ ( s t r i p s - p r o b l e m . v a r i a b l e s - o f ~ \Pi ) ~ v ) ) ~
            by auto
            hence is-cnf C
```
```
    by fastforce
    }
    thus ?thesis
    using is-cnf-BigAnd
    by blast
qed
private lemma encode-operator-effects-is-cnf-ii:
    is-cnf ( }\(map (\lambdav.\neg(Atom)(Operator t (index (strips-problem.operators-of \Pi)
op)))
    \vee ᄀ(Atom (State (Suc t) (index (strips-problem.variables-of \Pi) v)))) (delete-effects-of
op)))
proof -
    let ?fs = map ( \lambdav. \neg(Atom (Operator t (index (strips-problem.operators-of \Pi)
op)))
    \vee\neg(Atom (State (Suc t) (index (strips-problem.variables-of \Pi) v)))) (delete-effects-of
op)
    {
        fix C
        assume C\in set ?fs
        then obtain v
            where v\in set (delete-effects-of op)
                and C = ᄀ(Atom (Operator t (index (strips-problem.operators-of \Pi) op)))
                \vee \neg ( \text { Atom (State (Suc t) (index (strips-problem.variables-of П) v)))}
            by auto
        hence is-cnf C
            by fastforce
    }
    thus ?thesis
        using is-cnf-BigAnd
        by blast
qed
private lemma encode-operator-effect-is-cnf:
    shows is-cnf (encode-operator-effect \Pi top)
proof -
    let ?ops = strips-problem.operators-of \Pi
        and ?vs = strips-problem.variables-of \Pi
    let ?fs = map ( }\lambdav.\neg(\mathrm{ Atom (Operator t (index ?ops op)))
            \checkmark ~ A t o m ~ ( S t a t e ~ ( S u c ~ t ) ~ ( i n d e x ~ ? v s ~ v ) ) )
        (add-effects-of op)
        and ?fs' = map (\lambdav. }\neg(\mathrm{ Atom (Operator t (index ?ops op)))
                        \checkmark \neg ( \text { Atom (State (Suc t) (index ?vs v))))}
            (delete-effects-of op)
    have encode-operator-effect \Pi t op=\(?fs @ ?fs')
            unfolding encode-operator-effect-def[of \Pitop]
            by metis
    moreover {
            have }\forallf\in\mathrm{ set ?fs. is-cnf f }\forallf\in\mathrm{ set ?.fs'. is-cnf f
```
```
    using encode-operator-effects-is-cnf-i[of t П op]
            encode-operator-effects-is-cnf-ii[of t П op]
    by (simp+)
    hence }\forallf\in\operatorname{set}(?fs @ ?fs').is-cnf
    by auto
}
ultimately show ?thesis
    using is-cnf-BigAnd[of ?fs @ ?fs']
    by presburger
qed
private lemma set-map-encode-operator-effect[simp]:
    set (map ( }\lambda(t,op). encode-operator-effect \Pi t op) (List.product [0..<t]
        (strips-problem.operators-of П)))
    ={ encode-operator-effect \Pi k op
        | kop. (k,op) \in({0..<t} \times set (strips-problem.operators-of \Pi)) }
proof -
    let ?ops = strips-problem.operators-of \Pi
        and ?vs = strips-problem.variables-of \Pi
    let ?fs = map ( }\lambda(t,op). encode-operator-effect \Pit op) (List.product [0..<t] ?ops)
    have set ?fs = (\lambda(t,op). encode-operator-effect \Pitop)'({0..<t} × set ?ops)
        unfolding encode-operator-effect-def[of \Pi t]
        by force
    thus ?thesis
        using setcompr-eq-image[of \lambda(t,op). encode-operator-effect \Pi t op
            \lambda(k,op). (k,op) 
        by force
qed
private lemma encode-all-operator-effects-is-cnf:
    assumes is-valid-problem-strips \Pi
    shows is-cnf (encode-all-operator-effects \Pi (strips-problem.operators-of \Pi) t)
proof -
    let ?ops = strips-problem.operators-of \Pi
    let ?l = List.product [0..<t] ?ops
    let ?fs = map ( }\lambda(t,op). encode-operator-effect \Pit op) ?l l
    have }\forallf\in\mathrm{ set ?fs. is-cnf f
        using encode-operator-effect-is-cnf
        by force
    thus ?thesis
        unfolding encode-all-operator-effects-def
        using is-cnf-foldr-and-if[of ?fs]
        by presburger
qed
lemma encode-operators-is-cnf:
    assumes is-valid-problem-strips \Pi
    shows is-cnf (encode-operators \Pit)
```
```
unfolding encode-operators-def
```
using is-cnf-encode-all-operator-preconditions[of \(\Pi t]$
encode-all-operator-effects-is-cnf $[O F$ assms, of $t]$
is-cnf.simps(1)[of encode-all-operator-preconditions $\Pi$ (strips-problem.operators-of
П) $t$
encode-all-operator-effects $\Pi$ (strips-problem.operators-of $\Pi$ ) $t]$
by meson
- Simp flag alone did not do it, so we have to assign a name to this lemma as well.
private lemma set-map-to-operator-atom [simp]:
set (map ( (op. Atom (Operator $t$ (index (strips-problem.operators-of П) op)))
(filter ( $\lambda$ op. ListMem v vs) (strips-problem.operators-of П)))
$=\{$ Atom (Operator $t$ (index (strips-problem.operators-of П) op))
| op. op $\in$ set (strips-problem.operators-of $\Pi) \wedge v \in$ set vs $\}$
proof -
let ?ops $=$ strips-problem.operators-of $\Pi$
\{
have set (filter ( $\lambda o p$. ListMem $v$ vs) ?ops)
$=\{$ op $\in$ set ?ops. ListMem vvs $\}$
using set-filter
by force
then have set (filter ( $\lambda$ op. ListMem v vs) ?ops)
$=\{$ op. op $\in$ set ?ops $\wedge v \in$ set $v s\}$
using ListMem-iff [of $v$ ]
by blast
\}
then have set (map ( $\lambda$ op. Atom (Operator $t$ (index ?ops op)))
(filter ( $\lambda$ op. ListMem v vs) ?ops))
$=($ 入op. Atom $($ Operator $t$ (index ?ops op $))$ )' \{op $\in$ set ?ops. $v \in$ set vs $\}$
using set-map [of $\lambda$ op. Atom (Operator $t$ (index ?ops op))]
by presburger
thus ?thesis
by blast
qed
lemma is-disj-big-or-if:
assumes $\forall f \in$ set fs. is-lit-plus $f$
shows is-disj $\bigvee f s$
using assms
proof (induction $f_{s}$ )
case (Cons ffs)
have is-lit-plus $f$
using Cons.prems
by $\operatorname{simp}$
moreover have is-disj $\bigvee f s$
using Cons
by fastforce
ultimately show ?case
```    by simp qed simp lemma is-cnf-encode-negative-transition-frame-axiom:     shows is-cnf (encode-negative-transition-frame-axiom \Pitv) proof -     let ?vs = strips-problem.variables-of \Pi     and ?ops = strips-problem.operators-of \Pi     let ?deleting = filter (\lambdaop. ListMem v (delete-effects-of op)) ?ops     let ?fs = map (\lambdaop. Atom (Operator t (index ?ops op))) ?deleting         and ?A =(\neg(Atom (State t (index ?vs v))))         and ?B = Atom (State (Suc t) (index ?vs v))     {         fix f         assume f}\in\mathrm{ set ?fs         then obtain op             where op \in set ?ops                 and v\in set (delete-effects-of op)                 and f}=\mathrm{ Atom (Operator t (index ?ops op))             using set-map-to-operator-atom[of t \Pi v]             by fastforce     hence is-lit-plus f             by simp     } note nb=this     {         have is-disj \?fs             using is-disj-big-or-if nb             by blast         then have is-disj(?B \vee \?fs)             by force         then have is-disj (?A \vee (?B }\vee\vee?fs)             by fastforce         hence is-cnf (?A \vee (?B \vee \?fs))             by fastforce     }     thus ?thesis         unfolding encode-negative-transition-frame-axiom-def         by meson qed lemma is-cnf-encode-positive-transition-frame-axiom:     shows is-cnf (encode-positive-transition-frame-axiom \Pitv) proof -     let ?vs = strips-problem.variables-of \Pi         and ?ops = strips-problem.operators-of \Pi     let ?adding = filter (\lambdaop. ListMem v (add-effects-of op)) ?ops     let ?fs = map (\lambdaop. Atom (Operator t (index ?ops op))) ?adding         and ?A = Atom (State t (index ?vs v))```
```    and ?B = ᄀ(Atom (State (Suc t) (index ?vs v))) {     fix f     assume f}\in\mathrm{ set?fs     then obtain op     where op f set ?ops             and}v\in\mathrm{ set (add-effects-of op)             and f}=\mathrm{ Atom (Operator t(index ?ops op))     using set-map-to-operator-atom[of t \Piv]     by fastforce     hence is-lit-plus f     by simp } note nb=this {     have is-disj \?fs     using is-disj-big-or-if nb     by blast     then have is-disj(?B\vee \?fs)     by force     then have is-disj (?A \vee (?B \vee \?fs))         by fastforce     hence is-cnf (?A \vee (?B \vee \?fs))     by fastforce } thus ?thesis     unfolding encode-positive-transition-frame-axiom-def     by meson qed private lemma encode-all-frame-axioms-set[simp]:     set (map ( }\lambda(k,v). encode-negative-transition-frame-axiom \Pi kv             (List.product [0..<t] (strips-problem.variables-of \Pi))             @ (map (\lambda(k,v). encode-positive-transition-frame-axiom \Pi kv)             (List.product [0..<t] (strips-problem.variables-of \Pi))))     ={ encode-negative-transition-frame-axiom \Pikv             |kv. (k,v) \in ({0..<t} > set (strips-problem.variables-of \Pi)) }             { encode-positive-transition-frame-axiom \Pi kv             |kv.(k,v)\in({0..<t} \times set (strips-problem.variables-of \Pi)) } proof -     let ?l = List.product [0..<t] (strips-problem.variables-of \Pi)     let ?A = (\lambda(k,v). encode-negative-transition-frame-axiom \Pi k v)' set ?l         and ?B = (\lambda(k,v). encode-positive-transition-frame-axiom \Pi kv)'set ?l         and ?fs = map (\lambda(k,v). encode-negative-transition-frame-axiom \Pi kv)?l             @ (map (\lambda(k,v). encode-positive-transition-frame-axiom \Pi kv) ?l)         and ?vs = strips-problem.variables-of \Pi     have set-l-is: set ?l = {0..<t} }\times\mathrm{ set ?vs         by simp     have set ?fs = ?A \cup?B```
using set-append
by force
moreover have ? $A=\{$ encode-negative-transition-frame-axiom $\Pi k v$ $\mid k v .(k, v) \in(\{0 . .<t\} \times$ set ? $v s)\}$
using set-l-is setcompr-eq-image[of $\lambda(k, v)$. encode-negative-transition-frame-axiom
$\Pi k v$
$\lambda(k, v) .(k, v) \in(\{0 . .<t\} \times$ set ? $v s)]$
by fast
moreover have $? B=\{$ encode-positive-transition-frame-axiom $\Pi k v$ $\mid k v .(k, v) \in(\{0 . .<t\} \times$ set ? vs $)\}$
using set-l-is setcompr-eq-image[of $\lambda(k, v)$. encode-positive-transition-frame-axiom
$\Pi k v$
$\lambda(k, v) .(k, v) \in(\{0 . .<t\} \times$ set $? v s)]$
by fast
ultimately show ?thesis
by argo
qed
lemma encode-frame-axioms-is-cnf:
shows is-cnf (encode-all-frame-axioms $\Pi t$ )
proof -
let $? l=$ List.product $[0 . .<t]$ (strips-problem.variables-of $\Pi)$
and ? vs $=$ strips-problem.variables-of $\Pi$
let ? $A=\{$ encode-negative-transition-frame-axiom $\Pi k v$ $\mid k v .(k, v) \in(\{0 . .<t\} \times$ set ? $v s)\}$
and $? B=\{$ encode-positive-transition-frame-axiom $\Pi k v$
$\mid k v .(k, v) \in(\{0 . .<t\} \times$ set ? $v s)\}$
and ? $f_{s}=\operatorname{map}(\lambda(k, v)$. encode-negative-transition-frame-axiom $\Pi k v)$ ?l
@ (map $(\lambda(k, v)$. encode-positive-transition-frame-axiom $\Pi k v) ? l)$
\{
fix $f$
assume $f \in$ set ?fs
then consider ( $f$-encodes-negative-frame-axiom) $f \in ? A$
$\mid$ (f-encodes-positive-frame-axiom) $f \in$ ? $B$
by fastforce
hence $i s-c n f f$
using is-cnf-encode-negative-transition-frame-axiom
is-cnf-encode-positive-transition-frame-axiom
by (smt mem-Collect-eq)
\}
thus ?thesis
unfolding encode-all-frame-axioms-def
using is-cnf-BigAnd[of ?fs]
by meson
qed
lemma is-cnf-encode-problem:
```    assumes is-valid-problem-strips \Pi     shows is-cnf ( }\Phi\Pit proof -     have is-cnf ( }\mp@subsup{\Phi}{I}{}\Pi         using is-cnf-encode-initial-state assms         by auto     moreover have is-cnf (encode-goal-state \Pit)         using encode-goal-state-is-cnf[OF assms]         by simp     moreover have is-cnf (encode-operators \Pi t ^ encode-all-frame-axioms \Pit)         using encode-operators-is-cnf[OF assms] encode-frame-axioms-is-cnf         unfolding encode-transitions-def         by simp     ultimately show ?thesis         unfolding encode-problem-def SAT-Plan-Base.encode-problem-def         encode-transitions-def encode-initial-state-def[symmetric] encode-goal-state-def[symmetric]         by simp qed lemma encode-problem-has-model-then-also-partial-encodings:     assumes }\mathcal{A}\modelsSAT-Plan-Base.encode-problem \Pi     shows \mathcal{A}\modelsSAT-Plan-Base.encode-initial-state \Pi         and \mathcal{A}\modelsSAT-Plan-Base.encode-goal-state \Pit         and \mathcal{A}\modelsSAT-Plan-Base.encode-operators \Pit         and \mathcal{A}\modelsSAT-Plan-Base.encode-all-frame-axioms \Pit     using assms     unfolding SAT-Plan-Base.encode-problem-def     by simp+ lemma cnf-of-encode-problem-structure:     shows cnf (SAT-Plan-Base.encode-initial-state \Pi)         \subseteq c n f ~ ( S A T - P l a n - B a s e . e n c o d e - p r o b l e m ~ \Pi t )         and cnf (SAT-Plan-Base.encode-goal-state \Pi t)             \subseteq c n f ~ ( S A T - P l a n - B a s e . e n c o d e - p r o b l e m ~ \Pi ~ t ) ~         and cnf (SAT-Plan-Base.encode-operators \Pi t)             \subseteq c n f ( S A T - P l a n - B a s e . e n c o d e - p r o b l e m ~ \Pi t )         and cnf (SAT-Plan-Base.encode-all-frame-axioms \Pi t)             \subseteq c n f ( S A T - P l a n - B a s e . e n c o d e - p r o b l e m ~ \Pi t )     unfolding SAT-Plan-Base.encode-problem-def     SAT-Plan-Base.encode-problem-def[of \Pit] SAT-Plan-Base.encode-initial-state-def[of \Pi]     SAT-Plan-Base.encode-goal-state-def[of \Pit] SAT-Plan-Base.encode-operators-def     SAT-Plan-Base.encode-all-frame-axioms-def[of \Pit] subgoal by auto subgoal by force subgoal by auto subgoal by force done```
- A technical lemma which shows a simpler form of the CNF of the initial state encoding.
```private lemma cnf-of-encode-initial-state-set-i:     shows cnf ( }\mp@subsup{\Phi}{I}{}\Pi)=\bigcup{cnf (encode-state-variable 0         (index (strips-problem.variables-of \Pi) v) (((\Pi\mp@subsup{)}{I}{})v))             |v.v\in set (strips-problem.variables-of \Pi) ^((\Pi)I})v\not=None proof -     let ?vs = strips-problem.variables-of \Pi         and ?I = strips-problem.initial-of \Pi     let ?ls = map ( \lambdav. encode-state-variable 0 (index ?vs v) (?I v)\vee &)         (filter (\lambdav. ?I v\not= None) ?vs)     {         have cnf'set ?ls = cnf '( \lambdav. encode-state-variable 0 (index ?vs v) (?I v) V \perp)             ' set (filter (\lambdav. ?I v}=\mathrm{ None) ?vs)             using set-map[of \lambdav. encode-state-variable 0 (index ?vs v) (?I v) \vee \perp]             by presburger             also have ... = (\lambdav.cnf (encode-state-variable 0 (index ?vs v) (?I v)\vee &))                 ' set (filter ( }\lambdav.\mathrm{ ?I v}\not=\mathrm{ None) ?vs)             using image-comp             by blast     also have ... = (\lambdav.cnf (encode-state-variable 0 (index ?vs v) (?I v)))             ' {v\in set ?vs. ?I v}\not=\mathrm{ None }             using set-filter[of \lambdav. ?I v}\not=\mathrm{ None ?vs]             by auto     finally have cnf'set ?ls = { cnf (encode-state-variable 0 (index ?vs v) (?I v))             |v.v\in set ?vs ^ ?I v = None }             using setcompr-eq-image[of \lambdav.cnf (encode-state-variable 0 (index ?vs v) (?I v))]             by presburger     }     moreover have cnf ( }\mp@subsup{\Phi}{I}{}\Pi)=\bigcup(cnf'set ?ls         unfolding encode-initial-state-def SAT-Plan-Base.encode-initial-state-def         using cnf-BigAnd[of ?ls]         by meson     ultimately show ?thesis         by auto qed```
- A simplification lemma for the above one.
corollary cnf-of-encode-initial-state-set-ii:
assumes is-valid-problem-strips $\Pi$
shows cnf $\left(\Phi_{I} \Pi\right)=(\bigcup v \in$ set (strips-problem.variables-of $\Pi$ ). \{\{
literal-formula-to-literal (encode-state-variable 0 (index (strips-problem.variables-of
П) $v$ )
(strips-problem.initial-of $\Pi v)$ ) \}\})
proof -
```let ?vs = strips-problem.variables-of \Pi     and ?I = strips-problem.initial-of \Pi     have }n\mp@subsup{b}{1}{}:{v.v\in\mathrm{ set ?vs ^ ?I v}\not=\mathrm{ None } = set ?vs     using is-valid-problem-strips-initial-of-dom assms(1)     by auto {     fix }     assume v\in set ?vs     then have ?I v}\not=\mathrm{ None         using is-valid-problem-strips-initial-of-dom assms(1)         by auto     then consider (I-v-is-Some-True) ?I v = Some True         | (I-v-is-Some-False) ?I v = Some False         by fastforce     hence cnf (encode-state-variable 0 (index ?vs v) (?I v))         ={{ literal-formula-to-literal (encode-state-variable 0 (index ?vs v) (?I v)) }}         unfolding encode-state-variable-def         by (cases, simp+) } note nb {     have { cnf (encode-state-variable 0 (index ?vs v) (?I v))| v.v f set ?vs ^ ?I v}=\mathrm{ None }         = (\lambdav.cnf (encode-state-variable 0 (index ?vs v) (?I v)))'set ?vs     using setcompr-eq-image[of \lambdav.cnf (encode-state-variable 0 (index ?vs v) (?I v)) \lambdav.v\in set ?vs ^ ?I v\not=None] using n\mp@subsup{b}{1}{}     by presburger     hence { cnf (encode-state-variable 0 (index ?vs v) (?I v))|v.v\in set ?vs ^ ?I v}=\mathrm{ None }     = (\lambdav. {{ literal-formula-to-literal (encode-state-variable 0 (index ?vs v) (?I v)) }})             ' set ?vs             using n\mp@subsup{b}{2}{}             by force } thus ?thesis     using cnf-of-encode-initial-state-set-i     by (smt Collect-cong) qed lemma cnf-of-encode-initial-state-set:     assumes is-valid-problem-strips \Pi     and}v\in\operatorname{dom (strips-problem.initial-of \Pi) shows strips-problem.initial-of \Pi v=Some True \longrightarrow (\exists!C.C\incnf ( }\mp@subsup{\Phi}{I}{}\Pi     \wedge C = \{ ( S t a t e ~ 0 ~ ( i n d e x ~ ( s t r i p s - p r o b l e m . v a r i a b l e s - o f ~ \Pi ) ~ v ) ) + ~ \} )     and strips-problem.initial-of \Pi v = Some False \longrightarrow (\exists!C.C\incnf ( }\mp@subsup{\Phi}{I}{}\Pi         \wedge C = \{ ( S t a t e ~ 0 ~ ( i n d e x ~ ( s t r i p s - p r o b l e m . v a r i a b l e s - o f ~ \Pi ) ~ v ) ) ~ - 1 ~ \} )```
```proof -     let ?I = (\Pi) I     let ?vs = strips-problem.variables-of \Pi     let ? }\mp@subsup{\Phi}{I}{}=\mp@subsup{\Phi}{I}{}     have n\mp@subsup{b}{1}{}:cnf (\mp@subsup{\Phi}{I}{}\Pi)=\bigcup { cnf (encode-state-variable 0 (index ?vs v)             (strips-problem.initial-of \Pi v))|v.v\in set ?vs ^ ?I v\not= None }     using cnf-of-encode-initial-state-set-i     by blast     {     have v\in set ?vs             using is-valid-problem-strips-initial-of-dom assms(1, 2)             by blast     hence }v\in{v.v\in\mathrm{ set ?vs }\wedge\mathrm{ ?I v}\not=\mathrm{ None }             using assms(2)             by auto     } note n\mp@subsup{b}{2}{}= this     show strips-problem.initial-of \Pi v=Some True \longrightarrow (\exists!C.C\incnf (\Phi}\mp@subsup{\Phi}{I}{}\Pi         \wedge C = \{ ( S t a t e ~ 0 ~ ( i n d e x ~ ( s t r i p s - p r o b l e m . v a r i a b l e s - o f ~ \Pi ) ~ v ) ) + ~ \} )     and strips-problem.initial-of \Pi v=Some False \longrightarrow(\exists!C.C Ccnf ( }\mp@subsup{\Phi}{I}{}\Pi         \wedge C = \{ ( S t a t e ~ 0 ~ ( i n d e x ~ ( s t r i p s - p r o b l e m . v a r i a b l e s - o f ~ \Pi ) ~ v ) ) ~ - 1 ~ \} )         proof (auto)             assume i-v-is-some-true: strips-problem.initial-of \Pi v=Some True             then have {(State 0 (index (strips-problem.variables-of \Pi)v))+}                 \epsilonnf (encode-state-variable 0 (index (strips-problem.variables-of \Pi) v) (?I v))                 unfolding encode-state-variable-def                 using i-v-is-some-true                 by auto             thus {(State 0 (index (strips-problem.variables-of \Pi) v))+}                 cnf ( }\mp@subsup{\Phi}{I}{}\Pi                 using n\mp@subsup{b}{1}{}n\mp@subsup{b}{2}{}                 by auto     next         assume i-v-is-some-false: strips-problem.initial-of \Pi v=Some False         then have {(State 0 (index (strips-problem.variables-of \Pi)v))}\mp@subsup{)}{}{-1}                 \epsiloncnf (encode-state-variable 0 (index (strips-problem.variables-of \Pi) v) (?I v))                 unfolding encode-state-variable-def                 using i-v-is-some-false                 by auto     thus {(State 0 (index (strips-problem.variables-of \Pi) v) )}\mp@subsup{)}{}{-1}                 |nf ( }\mp@subsup{\Phi}{I}{}\Pi                 using n\mp@subsup{b}{1}{}}n\mp@subsup{b}{2}{                 by auto     qed qed lemma cnf-of-operator-encoding-structure:     cnf (encode-operators \Pit)=cnf (encode-all-operator-preconditions \Pi```
```    (strips-problem.operators-of П) t)     \cup \mathrm { cnf } ( \text { encode-all-operator-effects П (strips-problem.operators-of П) t)}     unfolding encode-operators-def     using cnf.simps(5)     by metis corollary cnf-of-operator-precondition-encoding-subset-encoding:     cnf (encode-all-operator-preconditions \Pi (strips-problem.operators-of \Pi) t)     \subseteq c n f ( \Phi \Pi t )     using cnf-of-operator-encoding-structure cnf-of-encode-problem-structure subset-trans     unfolding encode-problem-def     by blast lemma cnf-foldr-and[simp]:     cnf (foldr (^) fs (\neg\perp)) =(\bigcupf\in set fs.cnff) proof (induction fs)     case (Cons ffs)     have ih:cnf (foldr (^) fs (\neg\perp))}=(\bigcupf\in\mathrm{ set fs.cnff)         using Cons.IH         by blast     {         have cnf (foldr (\wedge) (f# fs ) (\neg\perp)) = cnf (f ^ foldr ( }\wedge)fs(\neg\perp)             by simp         also have ... = cnff \cupcnf (foldr ( }\wedge)fs(\neg\perp)             by force         finally have cnf (foldr ( }\wedge)(f#fs)(\neg\perp))=cnff\cup(\bigcupf\inset fs.cnff             using ih             by argo     }     thus ?case         by auto qed simp private lemma cnf-of-encode-operator-precondition[simp]:     cnf (encode-operator-precondition \Pitop) = (\bigcupv\in set (precondition-of op).     {{(Operator t (index (strips-problem.operators-of \Pi) op))}\mp@subsup{)}{}{-1             ,(State t (index (strips-problem.variables-of \Pi) v))+}}) proof -     let ?vs = strips-problem.variables-of \Pi         and ?ops = strips-problem.operators-of \Pi         and ?}\mp@subsup{\Phi}{P}{}=\mathrm{ encode-operator-precondition }\Pit\mathrm{ op     let ?fs = map (\lambdav. ᄀ (Atom (Operator t (index ?ops op))) \vee Atom (State t (index ?vs v)))     (precondition-of op)     and ?A = (\lambdav.\neg(Atom (Operator t (index ?ops op))) \vee Atom (State t (index ?vs v)))             ` set (precondition-of op)```
```    have cnf (encode-operator-precondition \Pitop) = cnf (\bigwedge?fs)     unfolding encode-operator-precondition-def     by presburger     also have ... = \ (cnf'set?fs)     using cnf-BigAnd     by blast     also have ... = \ (cnf'?A)     using set-map[of \lambdav.\neg (Atom (Operator t (index ?ops op))) \vee Atom (State t (index ?vs v))         precondition-of op]     by argo     also have ... = ( \bigcupv vet (precondition-of op).         cnf (\neg(Atom (Operator t (index ?ops op))) \vee Atom (State t (index ?vs v))))         by blast     finally show ?thesis     by auto qed lemma cnf-of-encode-all-operator-preconditions-structure[simp]:     cnf (encode-all-operator-preconditions \Pi (strips-problem.operators-of \Pi) t)     =(\bigcup(t,op) \in({..<t} \times set (operators-of \Pi)).         ( \bigcupv\in set (precondition-of op).             {{(Operator t (index (strips-problem.operators-of \Pi) op))}\mp@subsup{)}{}{-1                         ,(State t (index (strips-problem.variables-of \Pi) v))+}})) proof -     let ?vs = strips-problem.variables-of \Pi         and ?ops = strips-problem.operators-of \Pi     let ?l = List.product [0..<t] ?ops         and ?}\mp@subsup{\Phi}{P}{}=\mathrm{ encode-all-operator-preconditions П (strips-problem.operators-of П) t     let ?A = set (map ( }\lambda(t,op). encode-operator-precondition \Pit op) ?l)     {         have set ?l = {0..<t} }\times\mathrm{ set ((П)             by auto         then have ?A = (\lambda(t,op). encode-operator-precondition \Pi top)'({0..<t} } set ((\Pi)             using set-map             by force     } note nb = this     have cnf ? }\mp@subsup{\Phi}{P}{}=\operatorname{cnf}(foldr ( ^) (map ( \lambda(t,op). encode-operator-precondition \Pi top) ?l)(\neg\perp))             unfolding encode-all-operator-preconditions-def             by presburger     also have ...=(\f\in?A.cnff)         by simp     also have ... =(\bigcup(k,op) \in({0..<t} \times set ((\Pi\mp@subsup{)}{\mathcal{O}}{})).```
cnf (encode-operator-precondition $\Pi k$ op $)$ )
using $n b$
by fastforce
finally show ?thesis
by fastforce
qed
corollary cnf-of-encode-all-operator-preconditions-contains-clause-if:
fixes $\Pi$ ::'variable STRIPS-Representation.strips-problem
assumes is-valid-problem-strips ( $\Pi::$ 'variable STRIPS-Representation.strips-problem)
and $k<t$
and $o p \in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)$
and $v \in \operatorname{set}$ (precondition-of op)
shows \{ (Operator $k$ (index (strips-problem.operators-of П) op $))^{-1}$
, (State $k$ (index (strips-problem.variables-of $\Pi$ ) $v$ ) $\left.)^{+}\right\}$
$\in \operatorname{cnf}$ (encode-all-operator-preconditions $\Pi$ (strips-problem.operators-of П) t)
proof -
let ?ops $=$ strips-problem.operators-of $\Pi$
and ? $v s=$ strips-problem.variables-of $\Pi$
let ? $\Phi_{P}=$ encode-all-operator-preconditions $\Pi$ ?ops $t$ and $? C=\{(\text { Operator } k(\text { index (strips-problem.operators-of } \Pi) o p))^{-1}$
, (State $\left.k(\text { index (strips-problem.variables-of } \Pi \text { ) v) })^{+}\right\}$
$\{$
have $n b:(k, o p) \in\{. .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)$
using $\operatorname{assms}(2,3)$
by blast
moreover \{
have ? $C \in(\bigcup v \in$ set (precondition-of op).
$\left\{\{(\text { Operator } k \text { (index (strips-problem.operators-of } \Pi) \text { op) })^{-1}\right.$,
(State $k$ (index (strips-problem.variables-of $\Pi$ ) $\left.\left.v))^{+}\right\}\right\}$)
using $U N$-iff [where $A=$ set (precondition-of op) and $B=\lambda v$. $\left\{\{(\text { Operator } t \text { (index (strips-problem.operators-of } \Pi \text { ) } o p))^{-1}\right.$,
(State $t$ (index (strips-problem.variables-of П) v) $\left.\left.\left.)^{+}\right\}\right\}\right] \operatorname{assms}(4)$ by blast
hence $\exists x \in\{. .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)$.
$? C \in($ case $x$ of $(k, o p) \Rightarrow \bigcup v \in$ set (precondition-of op).
$\{\{(\text { Operator } k \text { (index (strips-problem.operators-of } \Pi) \text { op }))^{-1}$, (State $k$ (index (strips-problem.variables-of $\Pi$ ) $\left.\left.v))^{+}\right\}\right\}$)
using $n b$
by blast
\}
ultimately have $? C \in\left(\bigcup(t, o p) \in\left(\{. .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)\right)\right.$.
$(\bigcup v \in \operatorname{set}$ (precondition-of op).
$\left.\left.\left\{\left\{(\text { Operator } t(\text { index ?ops op }))^{-1},(\text { State } t(\text { index ?vs v }))^{+}\right\}\right\}\right)\right)$
by blast
\}
thus ?thesis
using cnf-of-encode-all-operator-preconditions-structure[of $\Pi t$ ]

## by argo <br> qed

corollary cnf-of-encode-all-operator-effects-subset-cnf-of-encode-problem:
cnf (encode-all-operator-effects $\Pi$ (strips-problem.operators-of П) t)
$\subseteq \operatorname{cnf}(\Phi \Pi t)$
using cnf-of-encode-problem-structure(3) cnf-of-operator-encoding-structure
unfolding encode-problem-def
by blast
private lemma cnf-of-encode-operator-effect-structure[simp]:
cnf (encode-operator-effect $\Pi t$ op)
$=(\bigcup v \in$ set (add-effects-of op). $\{\{$ (Operator $t$ (index (strips-problem.operators-of
П) $o p))^{-1}$
(State (Suc t) (index (strips-problem.variables-of $\Pi$ ) v) $\left.\left.\left.)^{+}\right\}\right\}\right)$
$\cup(\bigcup v \in$ set (delete-effects-of op).
$\left\{\left\{(\text { Operator } t \text { (index (strips-problem.operators-of } \Pi \text { ) op) })^{-1}\right.\right.$

$$
\text { , (State } \left.\left.\left.(\text { Suc t) }(\text { index }(\text { strips-problem.variables-of } \Pi) v))^{-1}\right\}\right\}\right)
$$

proof -
let ?fs $s_{1}=\operatorname{map}(\lambda v . \neg($ Atom $($ Operator $t($ index (strips-problem.operators-of $\Pi)$
$o p)$ ))
$\vee$ Atom (State (Suc t) (index (strips-problem.variables-of П) v)) )
(add-effects-of op)
and ?fs $s_{2}=\operatorname{map}(\lambda v . \neg($ Atom (Operator $t$ (index (strips-problem.operators-of
П) $o p)$ ))
$\vee \neg(\operatorname{Atom}($ State $($ Suc t) $)($ index $($ strips-problem.variables-of $\Pi) v))))$
(delete-effects-of op)

## \{

have cnf'set?fs $=c n f$
‘ $(\lambda v . \neg($ Atom $($ Operator $t($ index $($ strips-problem.operators-of $\Pi)$ op $)))$
$\vee$ Atom (State (Suc t) (index (strips-problem.variables-of П) v)))' set
(add-effects-of op)
using set-map
by force
also have $\ldots=(\lambda v . \operatorname{cnf}(\neg($ Atom (Operator $t$ (index (strips-problem.operators-of
П) $o p$ ))
$\vee \operatorname{Atom}($ State $($ Suc $t)($ index (strips-problem.variables-of $\Pi) v)))$
‘ set (add-effects-of op)
using image-comp
by blast
finally have $c n f$ 'set $? f s_{1}=(\lambda v .\{\{$ (Operator $t$ (index (strips-problem.operators-of
П) $o p))^{-1}$
, (State (Suc t) (index (strips-problem.variables-of $\Pi$ ) v)) $\left.\left.{ }^{+}\right\}\right\}$)'set (add-effects-of $o p)$
by auto
$\}$ note $n b_{1}=$ this
\{
have cnf'set ? ${ }^{\prime} s_{2}=c n f$ ' $(\lambda v . \neg($ Atom (Operator $t$ ( index (strips-problem.operators-of

```
П) }op))
        \vee \neg ( \text { Atom (State (Suc t) (index (strips-problem.variables-of П) v))))}
        ' set (delete-effects-of op)
        using set-map
        by force
    also have \ldots. = (\lambdav.cnf ( }\neg\mathrm{ (Atom (Operator t (index (strips-problem.operators-of
П) }op))
        \vee \neg ( \text { Atom (State (Suc t) (index (strips-problem.variables-of П) v)))))}
        'set (delete-effects-of op)
        using image-comp
        by blast
    finally have cnf'set ?fs s = (\lambdav. {{ (Operator t (index (strips-problem.operators-of
П) op)\mp@subsup{)}{}{-1}
        ,(State (Suc t) (index (strips-problem.variables-of \Pi) v))}\mp@subsup{)}{}{-1}}}
            ' set (delete-effects-of op)
        by auto
    } note nb 
    {
        have cnf(encode-operator-effect \Pit op)=\bigcup(cnf'set (?fs⿱丶万⿱⿰㇒一十凵
        unfolding encode-operator-effect-def
        using cnf-BigAnd[of ?fs ( @ ?fs
        by meson
    also have ... = \bigcup(cnf'set ?fs\mp@subsup{s}{1}{}\cupcnf'set?fs\mp@subsup{s}{2}{})
        using set-append[of ?fs\mp@subsup{s}{1}{}?f\mp@subsup{s}{2}{}] image-Un[of cnf set?fs set ?fs}\mp@subsup{s}{2}{}
        by argo
    also have ... = \bigcup (cnf'set ?fs s) \cup \bigcup(cnf' set ?fs s)
        using Union-Un-distrib[of cnf' set ?fs cnf 'set ?fs ( ]
        by argo
    finally have cnf (encode-operator-effect \Pi t op)
            = (\bigcupv set (add-effects-of op).
            {{(Operator t (index (strips-problem.operators-of \Pi) op))}\mp@subsup{)}{}{-1
                    , (State (Suc t) (index (strips-problem.variables-of \Pi) v))+ }})
            \cup ( \bigcup v \in \operatorname { s e t ~ ( d e l e t e - e f f e c t s - o f ~ o p ) . }
                    {{(Operator t (index (strips-problem.operators-of \Pi) op))}\mp@subsup{)}{}{-1
                    , (State (Suc t) (index (strips-problem.variables-of \Pi) v))}\mp@subsup{)}{}{-1}}}
        using n\mp@subsup{b}{1}{}n\mp@subsup{b}{2}{}
        by argo
}
thus ?thesis
    by blast
qed
lemma cnf-of-encode-all-operator-effects-structure:
    cnf (encode-all-operator-effects \Pi (strips-problem.operators-of \Pi) t)
    =(\bigcup(k,op) \in({0..<t} \times set ((\Pi\mp@subsup{)}{\mathcal{O}}{)})).
        (\bigcupv\in set (add-effects-of op).
        {{(Operator k (index (strips-problem.operators-of \Pi) op))}\mp@subsup{)}{}{-1
```

```
            ,(State (Suc k) (index (strips-problem.variables-of \Pi) v))+ }}))
    \cup ( \bigcup ( k , o p ) \in ( \{ 0 . . < t \} \times \operatorname { s e t } ( ( \Pi ) _ { \mathcal { O } } ) ) .
    ( \bigcupv vet (delete-effects-of op).
        {{(Operator k (index (strips-problem.operators-of \Pi) op))}\mp@subsup{)}{}{-1
            , (State (Suc k) (index (strips-problem.variables-of \Pi) v))}\mp@subsup{)}{}{-1}}})
proof -
    let ?ops = strips-problem.operators-of \Pi
        and ?vs = strips-problem.variables-of \Pi
    let ? }\mp@subsup{\Phi}{E}{}=\mathrm{ encode-all-operator-effects }\Pi\mathrm{ ?ops t
        and ?l = List.product [0..<t] ?ops
```



```
    have nb: set (List.product [0..<t] ?ops) = {0..<t} }\times\mathrm{ set ?ops
        by simp
    {
        have cnf'set ?fs = cnf' ( }\lambda(k,op). encode-operator-effect \Pi kop ''({0..<t
* set ?ops)
            by force
            also have ... = (\lambda(k,op).cnf (encode-operator-effect \Pi kop))'({0..<t} }
set ?ops)
            using image-comp
            by fast
        finally have cnf'set ?fs = ( }\lambda(k,op)
                    ( }\bigcupv\in\mathrm{ set (add-effects-of op).
                    {{(Operator k (index (strips-problem.operators-of \Pi) op))}\mp@subsup{)}{}{-1
                        , (State (Suc k) (index (strips-problem.variables-of \Pi) v))+}}}
                \cup ( \bigcup v \in ~ s e t ~ ( d e l e t e - e f f e c t s - o f ~ o p ) .
                    {{(Operator k (index (strips-problem.operators-of \Pi) op))}\mp@subsup{)}{}{-1
                    , (State (Suc k) (index (strips-problem.variables-of \Pi) v))}\mp@subsup{)}{}{-1}}})
            '({0..<t} > set ?ops )
            using cnf-of-encode-operator-effect-structure
            by auto
    }
    thus ?thesis
        unfolding encode-all-operator-effects-def
        using cnf-BigAnd[of ?fs]
        by auto
qed
corollary cnf-of-operator-effect-encoding-contains-add-effect-clause-if:
    fixes П:: 'a strips-problem
    assumes is-valid-problem-strips \Pi
        and k<t
        and op }\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{}
        and}v\in\mathrm{ set (add-effects-of op)
    shows {(Operator k (index (strips-problem.operators-of \Pi) op))}\mp@subsup{)}{}{-1
        , (State (Suc k) (index (strips-problem.variables-of \Pi) v))+}
    cnf (encode-all-operator-effects \Pi (strips-problem.operators-of \Pi) t)
```

```
proof -
    let ? }\mp@subsup{\Phi}{E}{}=\mathrm{ encode-all-operator-effects П (strips-problem.operators-of П)t
        and ?ops = strips-problem.operators-of \Pi
        and ?vs = strips-problem.variables-of \Pi
    let ?Add = \bigcup(k,op)\in{0..<t} \times set ((\Pi\mp@subsup{)}{\mathcal{O}}{})\mathrm{ .}
        \veset (add-effects-of op). {{ (Operator k (index ?ops op))}\mp@subsup{)}{}{-1}\mathrm{ , (State (Suc k)
(index ?vs v))+}}
    let ?C = {(Operator k (index ?ops op) )}\mp@subsup{)}{}{-1},(\mathrm{ State (Suc k) (index ?vs v))+}
    have ?Add\subseteqcnf ? }\mp@subsup{\Phi}{E}{
        using cnf-of-encode-all-operator-effects-structure[of \Pi t] Un-upper1[of ?Add]
        by presburger
    moreover {
    have ?C \in{{(Operator k (index ?ops op))}\mp@subsup{)}{}{-1}\mathrm{ ,(State (Suc k) (index ?vs v))+
}}
                using assms(4)
                by blast
        then have ?C C\in( \bigcupv\inset (add-effects-of op).
                {{(Operator k (index ?ops op))}\mp@subsup{)}{}{-1},(\mathrm{ State (Suc k) (index ?vs v))}\mp@subsup{)}{}{+}}}
            using Complete-Lattices.UN-iff[of ?C \lambdav.{{(Operator k (index ?ops op))}\mp@subsup{)}{}{-1
                    ,(State (Suc k) (index ?vs v))}\mp@subsup{)}{}{+}}}\mathrm{ set (add-effects-of op)]
        using assms(4)
        by blast
    moreover have (k,op) \in({0..<t} \times set ((\Pi)
            using assms(2, 3)
            by fastforce
    ultimately have ?C }\in\mathrm{ ?Add
            by blast
    }
    ultimately show ?thesis
        using subset-eq[of ?Add cnf ?'\Phi
        by meson
qed
corollary cnf-of-operator-effect-encoding-contains-delete-effect-clause-if:
    fixes \Pi:: 'a strips-problem
    assumes is-valid-problem-strips \Pi
    and k<t
    and op }\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{)
    and}v\in\mathrm{ set (delete-effects-of op)
    shows {(Operator k (index (strips-problem.operators-of \Pi) op))}\mp@subsup{)}{}{-1
            , (State (Suc k) (index (strips-problem.variables-of \Pi) v)}\mp@subsup{)}{}{-1}
    cnf (encode-all-operator-effects \Pi (strips-problem.operators-of \Pi) t)
proof -
    let ? }\mp@subsup{\Phi}{E}{}=\mathrm{ encode-all-operator-effects П (strips-problem.operators-of П) t
        and ?ops = strips-problem.operators-of \Pi
    and ?vs = strips-problem.variables-of \Pi
    let ?Delete = (U(k,op) ){0..<t} \times set ((\Pi)
    U\inset (delete-effects-of op).
```

```
    {{(Operator k (index ?ops op))}\mp@subsup{)}{}{-1},(\mathrm{ State (Suc k) (index ?vs v))}\mp@subsup{)}{}{-1}}}
    let ?C = {(Operator k (index ?ops op) )}\mp@subsup{)}{}{-1},(\mathrm{ State (Suc k) (index ?vs v)}\mp@subsup{)}{}{-1}
    have ?Delete }\subseteqcnf ?.\mp@subsup{\Phi}{E}{
    using cnf-of-encode-all-operator-effects-structure[of \Pit] Un-upper2[of ?Delete]
    by presburger
    moreover {
    have ?C }\in(\bigcupv\in\mathrm{ set (delete-effects-of op).
        {{(Operator k (index ?ops op))}\mp@subsup{)}{}{-1}\mathrm{ ,(State (Suc k) (index ?vs v))}\mp@subsup{)}{}{-1}}}
        using assms(4)
        by blast
    moreover have (k,op) \in{0..<t} \times set ?ops
    using assms(2, 3)
    by force
    ultimately have ?C }\in\mathrm{ ?Delete
        by fastforce
    }
    ultimately show ?thesis
    using subset-eq[of ?Delete cnf ?\Phi}\mp@subsup{\Phi}{E}{}
    by meson
qed
private lemma cnf-of-big-or-of-literal-formulas-is[simp]:
    assumes }\forallf\in\mathrm{ set fs. is-literal-formula }
    shows cnf (\bigveefs)={{ literal-formula-to-literal f|f.f\in set fs }}
    using assms
proof (induction fs)
    case (Cons ffs)
    {
        have is-literal-formula-f: is-literal-formula f
            using Cons.prems(1)
            by simp
    then have cnff={{ literal-formula-to-literal f }}
            using cnf-of-literal-formula
            by blast
    } note nb
    {
        have }\forall\mp@subsup{f}{}{\prime}\in\mathrm{ set fs. is-literal-formula }\mp@subsup{f}{}{\prime
            using Cons.prems
            by fastforce
        hence cnf (\ \s) = {{ literal-formula-to-literal f |f.f\in set fs }}
            using Cons.IH
            by argo
    } note n\mp@subsup{b}{2}{}= this
    {
        have cnf (\ (f# fs)) = (\lambda(g,h).g\cuph)
            '({{ literal-formula-to-literal f}}
```

```
            \times{{ literal-formula-to-literal f'| | f'. f' }\in\mathrm{ set fs }})
        using nb n nb 
        by simp
    also have ... = {{ literal-formula-to-literal f }
        \cup{literal-formula-to-literal f}\mp@subsup{f}{}{\prime}|\mp@subsup{f}{}{\prime}.\mp@subsup{f}{}{\prime}\in\mathrm{ set fs }}
        by fast
    finally have cnf (V(f#fs))={{ literal-formula-to-literal f'| f'. f' \in set (f
# fs) }}
        by fastforce
}
thus ?case.
qed simp
private lemma set-filter-op-list-mem-vs[simp]:
    set (filter (\lambdaop. ListMem v vs) ops) ={op.op\in set ops }\wedgev\in\mathrm{ set vs }
    using set-filter[of \lambdaop. ListMem v vs ops] ListMem-iff
    by force
private lemma cnf-of-positive-transition-frame-axiom:
    cnf (encode-positive-transition-frame-axiom \Pikv)
    ={{(State k (index (strips-problem.variables-of \Pi) v))}\mp@subsup{)}{}{+
        , (State (Suc k) (index (strips-problem.variables-of \Pi) v))}\mp@subsup{)}{}{-1}
        \cup {(Operator k (index (strips-problem.operators-of \Pi) op))+
            |op.op\in set (strips-problem.operators-of \Pi) }\wedgev\in set (add-effects-of op
}}
proof -
    let ?vs = strips-problem.variables-of \Pi
        and ?ops = strips-problem.operators-of \Pi
    let ?adding-operators = filter (\lambdaop. ListMem v (add-effects-of op)) ?ops
    let ?fs = map (\lambdaop. Atom (Operator k (index ?ops op))) ?adding-operators
    {
    have set ?fs = (\lambdaop. Atom (Operator k (index ?ops op)))' set ?adding-operators
            using set-map[of \lambdaop. Atom (Operator k (index ?ops op)) ?adding-operators]
            by blast
        then have literal-formula-to-literal'set ?fs
            = (\lambdaop. (Operator k (index ?ops op))+)' set ?adding-operators
            using image-comp[of literal-formula-to-literal \lambdaop. Atom (Operator k (index
        ?ops op))
            set ?adding-operators]
        by simp
        also have ... = (\lambdaop. (Operator k (index ?ops op))+}
            '{ op.op }\in\mathrm{ set ?ops }\wedgev\in\mathrm{ set (add-effects-of op) }
            using set-filter-op-list-mem-vs[of v - ?ops]
            by auto
        finally have literal-formula-to-literal'set ?fs
            ={(Operator k (index ?ops op))+}| op.op set?ops \wedgev\in set (add-effects-of
op) }
```

using setcompr-eq-image[of $\lambda$ op. (Operator $k($ index ?ops op $))^{+}$ $\lambda o p . o p \in s e t$ ?adding-operators]
by blast

```
    hence cnf (\?fs) = {{(Operator k(index ?ops op))+
    |op.op set ?ops }\wedgev\in\mathrm{ set (add-effects-of op) }}
    using cnf-of-big-or-of-literal-formulas-is[of ?fs]
        setcompr-eq-image[of literal-formula-to-literal }\lambdaf.f\in\mathrm{ set ?fs]
    by force
}
```

then have cnf $(\neg($ Atom $($ State $($ Suc $k)($ index ?vs $v))) \vee \bigvee ? f s)$
$=\left\{\left\{\left(\right.\right.\right.$ State $($ Suc $\left.k)(\text { index ?vs v) })^{-1}\right\} \cup\left\{(\text { Operator } k(\text { index ?ops op }))^{+}\right.$ $\mid$ op. op $\in$ set ?ops $\wedge v \in$ set (add-effects-of op) $\}\}$
by force
then have cnf $(($ Atom (State $k($ index ? vs v) $) \vee(\neg($ Atom (State (Suc k) (index ? vs $v))$ ) $\vee \bigvee ?$ ?fs $)$ )
$=\left\{\left\{(\text { State } k(\text { index ? } \text { ?s } v))^{+}\right\}\right.$
$\cup\left\{(\text { State }(\text { Suc } k)(\text { index ? ?s v } v))^{-1}\right\}$
$\cup\left\{(\text { Operator } k(\text { index ?ops op }))^{+} \mid\right.$op. op $\in$ set ?ops $\wedge v \in$ set (add-effects-of
op) $\}$ \}
by $\operatorname{simp}$
moreover have cnf (encode-positive-transition-frame-axiom $\Pi k v$ )
$=\operatorname{cnf}(($ Atom (State $k($ index ?vs $v)) \vee(\neg($ Atom (State (Suc $k)($ index ?vs v)))
$\vee \bigvee ? f s)))$
unfolding encode-positive-transition-frame-axiom-def
by metis
ultimately show ?thesis
by blast
qed
private lemma cnf-of-negative-transition-frame-axiom:
cnf (encode-negative-transition-frame-axiom $\Pi k v$ )
$=\left\{\left\{\left(\right.\right.\right.$ State $k(\text { index (strips-problem.variables-of П) v) })^{-1}$
, (State (Suc k) (index (strips-problem.variables-of П) v) $\left.)^{+}\right\}$
$\cup\left\{(\text { Operator } k \text { (index (strips-problem.operators-of П) op) })^{+}\right.$
| op. op $\in$ set (strips-problem.operators-of $\Pi$ ) $\wedge v \in$ set (delete-effects-of op)
\}\}
proof -
let ?vs $=$ strips-problem.variables-of $\Pi$
and ?ops $=$ strips-problem.operators-of $\Pi$
let ?deleting-operators $=$ filter $(\lambda o p . L i s t M e m v($ delete-effects-of op $))$ ?ops
let ?fs $=$ map $(\lambda o p . A t o m($ Operator $k($ index ?ops op $))$ ) ?deleting-operators
\{
have set?fs $=(\lambda o p$. Atom $($ Operator $k($ index ?ops op $)))$ ' set ?deleting-operators using set-map[of $\lambda$ op. Atom (Operator $k$ (index ?ops op)) ?deleting-operators]
by blast
then have literal-formula-to-literal'set ?fs
$=\left(\lambda\right.$ op. $\left.(\text { Operator } k(\text { index ?ops op }))^{+}\right)$'set ?deleting-operators
using image-comp[of literal-formula-to-literal $\lambda$ op. Atom (Operator $k$ (index ?ops op))
set?deleting-operators]
by $\operatorname{simp}$
also have $\ldots=\left(\lambda o p .(\text { Operator } k(\text { index ?ops op }))^{+}\right)$
' \{ op. op $\in$ set ?ops $\wedge v \in$ set (delete-effects-of op) \}
using set-filter-op-list-mem-vs[of v-?ops]
by auto
finally have literal-formula-to-literal' set ?fs
$=\left\{(\text { Operator } k(\text { index ?ops op }))^{+} \mid\right.$op. op $\in$ set ?ops $\wedge v \in$ set (delete-effects-of op) $\}$
using setcompr-eq-image $[\text { of } \lambda o p \text {. (Operator } k(\text { index ?ops op }))^{+}$
$\lambda o p$. op $\in$ set ?deleting-operators]
by blast
hence $\mathrm{cnf}(\mathrm{V}$ ?fs $)=\left\{\left\{(\text { Operator } k(\text { index ? ops op }))^{+}\right.\right.$
$\mid$ op. op $\in$ set ?ops $\wedge v \in$ set (delete-effects-of op) \}\}
using cnf-of-big-or-of-literal-formulas-is[of ?fs]
setcompr-eq-image[of literal-formula-to-literal $\lambda f . f \in$ set ?fs]
by force
\}
then have cnf (Atom (State (Suc k) (index ?vs v)) $\vee \bigvee$ ?fs)
$=\left\{\left\{(\text { State }(\text { Suc } k)(\text { index ?vs } v))^{+}\right\} \cup\left\{(\text { Operator } k \text { (index ?ops op) })^{+}\right.\right.$
$\mid$ op. op $\in$ set ?ops $\wedge v \in$ set (delete-effects-of op) $\}\}$
by force
then have cnf $\left(\left(\neg\left(\right.\right.\right.$ Atom $\left(\right.$ State $k\left(\right.$ index ? ${ }^{2}$ vs $\left.\left.\left.v\right)\right)\right) \vee($ Atom (State (Suc $k)$ (index ?vs $v)$ ) $\vee \bigvee$ ? fs $s))$ )
$=\left\{\left\{\left(\right.\right.\right.$ State $\left.k(\text { index ?vs v) })^{-1}\right\}$
$\cup\left\{(\text { State }(\text { Suc } k)(\text { index ?vs } v))^{+}\right\}$
$\cup\left\{(\text { Operator } k(\text { index ? ops op }))^{+} \mid\right.$op. op $\in$ set ?ops $\wedge v \in$ set (delete-effects-of $o p)\}\}$
by $\operatorname{simp}$
moreover have cnf (encode-negative-transition-frame-axiom $\Pi k v$ )
$=\operatorname{cnf}((\neg($ Atom (State $k$ (index ?vs $v))) \vee($ Atom (State (Suc k) (index ?vs
v)) $\vee \bigvee$ ? $f s))$ )
unfolding encode-negative-transition-frame-axiom-def
by metis
ultimately show ?thesis
by blast
qed

```
lemma cnf-of-encode-all-frame-axioms-structure:
    cnf (encode-all-frame-axioms \(\Pi t\) )
    \(=\bigcup\left(\bigcup(k, v) \in\left(\{0 . .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\right)\right.\).
        \{\{\{ (State \(k\) (index (strips-problem.variables-of \(\Pi\) ) v)) \({ }^{+}\)
                , (State (Suc k) (index (strips-problem.variables-of П) v) \(\left.)^{-1}\right\}\)
            \(\cup\left\{(\text { Operator } k(\text { index (strips-problem.operators-of } \Pi) \text { op) })^{+}\right.\)
                \(\mid\) op. op \(\in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right) \wedge v \in \operatorname{set}(\) add-effects-of op) \(\left.\left.\left.\}\right\}\right\}\right)\)
    \(\cup \bigcup\left(\bigcup(k, v) \in\left(\{0 . .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\right)\right.\).
    \{\{\{ (State \(k\) (index (strips-problem.variables-of П) v) \()^{-1}\)
                , (State (Suc k) (index (strips-problem.variables-of П) v) \(\left.)^{+}\right\}\)
                    \(\cup\left\{(\text { Operator } k(\text { index }(\text { strips-problem.operators-of } \Pi) \text { op }))^{+}\right.\)
                \(\mid\) op. op \(\in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right) \wedge v \in \operatorname{set}(\) delete-effects-of op) \(\left.\left.\left.\}\right\}\right\}\right)\)
proof -
    let ?vs = strips-problem.variables-of \(\Pi\)
        and ?ops \(=\) strips-problem.operators-of \(\Pi\)
        and \(? \Phi_{F}=\) encode-all-frame-axioms \(\Pi t\)
    let \(? l=\) List.product \([0 . .<t]\) ?vs
    let ?fs \(=\operatorname{map}(\lambda(k, v)\). encode-negative-transition-frame-axiom \(\Pi k v)\) ?l
        @ map \((\lambda(k, v)\). encode-positive-transition-frame-axiom \(\Pi k v)\) ?l
    \{
    let ? \(A=\{\) encode-negative-transition-frame-axiom \(\Pi k v\)
            \(\left.\mid k v .(k, v) \in\left(\{0 . .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\right)\right\}\)
            and \(? B=\{\) encode-positive-transition-frame-axiom \(\Pi k v\)
            \(\left.\mid k v .(k, v) \in\left(\{0 . .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\right)\right\}\)
    have set-l: set ?l \(=\{. .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\)
    using set-product
    by force
    have set ?fs \(=? A \cup ? B\)
    unfolding set-append set-map
    using encode-all-frame-axioms-set
    by force
    then have cnf'set ? \(s s=c n f\) ' ? \(A \cup c n f\) '? \(B\)
        using image-Un[of cnf ? A ?B]
        by argo
    moreover \{
    have \(? A=(\bigcup(k, v) \in(\{0 . .<t\} \times \operatorname{set}((\Pi) \mathcal{V}))\).
                \{ encode-negative-transition-frame-axiom \(\Pi k v\})\)
        by blast
    then have cnf'? \(A=(\bigcup(k, v) \in(\{0 . .<t\} \times \operatorname{set}((\Pi) \mathcal{V}))\).
                \(\{\operatorname{cnf}\) (encode-negative-transition-frame-axiom \(\Pi k v)\}\) )
                by blast
    hence cnf' \(? A=\left(\bigcup(k, v) \in\left(\{0 . .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\right)\right.\).
                \(\left\{\left\{\left\{(\text { State } k \text { (index ? vs v) })^{-1}\right.\right.\right.\)
                    , (State (Suc k) (index ?vs v) \(\left.)^{+}\right\}\)
                    \(\cup\left\{(\text { Operator } k(\text { index ?ops op }))^{+}\right.\)
                            \(\mid\) op. op \(\in\) set ? ops \(\wedge v \in\) set (delete-effects-of op) \}\}\})
                using cnf-of-negative-transition-frame-axiom[of \(\Pi\) ]
```

```
        by presburger
}
moreover {
    have ?B}=(\bigcup(k,v)\in({0..<t}\times set ((\Pi)\mathcal{V}))
        { encode-positive-transition-frame-axiom \Pikv})
        by blast
    then have cnf '?B = (U(k,v)\in({0...<t} \times set ((\Pi)\mathcal{V})).
        {cnf(encode-positive-transition-frame-axiom \Pikv) })
        by blast
    hence cnf '?B = (U(k,v) \in({0..<t} \times set ((\Pi)\mathcal{V})).
        {{{ (State k (index ?vs v))+
            , (State (Suc k) (index ?vs v))}\mp@subsup{)}{}{-1}
            \cup(Operator k (index ?ops op) )}\mp@subsup{)}{}{+
            | op.op { set ?ops }\wedgev\in\mathrm{ set (add-effects-of op) }}})
        using cnf-of-positive-transition-frame-axiom[of \Pi]
        by presburger
    }
    ultimately have cnf'set ?fs
    =(U(k,v) \in({0..<t} \times set ((\Pi)v)).
        {{{(State k (index ?vs v))+
            ,(State (Suc k) (index ?vs v))}\mp@subsup{)}{}{-1}
            \cup \{ ( \text { Operator k (index ?ops op))} ) ^ { + }
                |op.op\in set ((\Pi\mp@subsup{)}{\mathcal{O}}{})\wedgev\in\operatorname{set (add-effects-of op) }}})}
    \cup ( \bigcup ( k , v ) \in ( \{ 0 . . < t \} \times \operatorname { s e t } ( ( \Pi ) \mathcal { V } ) ) .
        {{{(State k (index ?vs v))}\mp@subsup{)}{}{-1
            , (State (Suc k) (index ?vs v))+ }
            \cup \{ ( \text { Operator k (index ?ops op) )} ) ^ { + }
                |op.op\in set ((\Pi\mp@subsup{)}{\mathcal{O}}{})\wedgev\in\operatorname{set}(\mathrm{ delete-effects-of op)}}})}
    unfolding set-append set-map
    by force
}
then have cnf (encode-all-frame-axioms \Pit)
    = U((U(k,v) \in({0..<t} \times set ((\Pi)\mathcal{V})).
        {{{(State k (index ?vs v))+
            , (State (Suc k) (index ?vs v))}\mp@subsup{)}{}{-1}
            \cup \{ ( \text { Operator k (index ?ops op))} ) ^ { + }
                |op.op }\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{\prime})\wedgev\in\operatorname{set}(add-effects-of op) }}})
    \cup(U(k,v)\in({0..<t} \times set ((\Pi)\mathcal{V})).
        {{{(State k (index ?vs v))}\mp@subsup{)}{}{-1
            , (State (Suc k) (index ?vs v))+}
            \cup \{ ( \text { Operator k (index ?ops op))} ) ^ { + }
            |op.op\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{})\wedgev\in\operatorname{set}(\mathrm{ delete-effects-of op)}}}))}
unfolding encode-all-frame-axioms-def Let-def
using cnf-BigAnd[of ?fs]
by argo
thus ?thesis
    using Union-Un-distrib[of
        (U(k,v) \in ({0..<t} × set ((\Pi)\mathcal{V})).
```

```
    \{\{\{ (State \(k\left(\right.\) index ? \({ }^{2}\) vs \(\left.\left.v\right)\right)^{+}\)
    , (State (Suc k) (index ?vs v) \(\left.)^{-1}\right\}\)
    \(\cup\left\{(\text { Operator } k(\text { index ?ops op }))^{+}\right.\)
        \(\mid\) op. op \(\in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right) \wedge v \in\) set (add-effects-of op) \(\left.\left.\left.\}\right\}\right\}\right)\)
    \(\left(\bigcup(k, v) \in\left(\{0 . .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\right)\right.\).
    \{\{\{ (State \(k\) (index ?vs v) \()^{-1}\)
        , (State (Suc k) (index ?vs v) \(\left.)^{+}\right\}\)
        \(\cup\left\{(\text { Operator } k(\text { index ?ops op }))^{+}\right.\)
        \(\mid\) op. op \(\in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right) \wedge v \in \operatorname{set}(\) delete-effects-of op \(\left.\left.\left.\left.\left.)\right\}\right\}\right\}\right)\right]\)
    by argo
qed
- A technical lemma used in .
private lemma cnf-of-encode-goal-state-set-i:
    cnf \(\left(\left(\Phi_{G} \Pi\right) t\right)=\bigcup(\{\) cnf (encode-state-variable \(t\)
    (index (strips-problem.variables-of \(\left.\Pi) v)\left(\left((\Pi)_{G}\right) v\right)\right)\)
    \(\mid v . v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right) \wedge\left((\Pi)_{G}\right) v \neq\) None \(\left.\}\right)\)
proof -
    let ?vs \(=\) strips-problem.variables-of \(\Pi\)
        and ? \(G=(\Pi)_{G}\)
        and \(? \Phi_{G}=\left(\Phi_{G} \Pi\right) t\)
    let ?fs \(=\operatorname{map}(\lambda v\). encode-state-variable \(t(\) index ? \(v s\) v) \((? G v) \vee \perp)\)
            (filter \((\lambda v . ? G v \neq\) None) ? \(v s)\)
    \{
        have \(c n f\) 'set ?fs \(=c n f\) ' \((\lambda v\). encode-state-variable \(t(\) index ?vs \(v)(? G v) \vee\)
」)
            ' \(\{v \mid v . v \in\) set ? vs \(\wedge ? G v \neq\) None \(\}\)
            unfolding set-map
            by force
        also have \(\ldots=(\lambda v\). cnf \((\) encode-state-variable \(t(\) index ? vs \(v)(? G v) \vee \perp))\)
            ' \(\{v \mid v . v \in\) set ? vs \(\wedge\) ? \(G v \neq\) None \(\}\)
            using image-comp[of cnf ( \(\lambda v\). encode-state-variable \(t\) (index ?vs v) (?G v) \(\vee\)
」)
            \(\{v \mid v . v \in\) set ?vs \(\wedge ? G v \neq\) None \(\}]\)
            by fast
    finally have cnf'set ?fs \(=\{\operatorname{cnf}(\) encode-state-variable \(t(\) index ?vs \(v)(? G v))\)
                \(\mid v . v \in\) set ? vs \(\wedge\) ? \(G v \neq\) None \(\}\)
            unfolding setcompr-eq-image[of \(\lambda v\). cnf (encode-state-variable \(t\) (index ?vs
\(v)(? G v) \vee \perp)]\)
            by auto
    \}
    moreover have \(\operatorname{cnf}\left(\left(\Phi_{G} \Pi\right) t\right)=\bigcup(c n f\) 'set ?fs \()\)
        unfolding encode-goal-state-def SAT-Plan-Base.encode-goal-state-def Let-def
        using cnf-BigAnd[of ? \(f s\) ]
        by force
    ultimately show?thesis
        by \(\operatorname{simp}\)
qed
```

- A simplification lemma for the above one.

```
corollary cnf-of-encode-goal-state-set-ii:
    assumes is-valid-problem-strips \(\Pi\)
    shows cnf \(\left(\left(\Phi_{G} \Pi\right) t\right)=\bigcup(\{\{\{\) literal-formula-to-literal
        (encode-state-variable \(t\) (index (strips-problem.variables-of \(\Pi\) ) \(\left.v)\left(\left((\Pi)_{G}\right) v\right)\right)\)
\}\}
    \(\mid v . v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right) \wedge\left((\Pi)_{G}\right) v \neq\) None \(\left.\}\right)\)
proof -
    let ?vs = strips-problem.variables-of \(\Pi\)
        and ? \(G=(\Pi)_{G}\)
        and \(? \Phi_{G}=\left(\Phi_{G} \Pi\right) t\)
    \{
        fix \(v\)
        assume \(v \in\left\{v \mid v . v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right) \wedge ? G v \neq\right.\) None \(\}\)
        then have \(v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\) and \(G\)-of-v-is-not-None: ? \(G v \neq\) None
        by fast+
    then consider \((A)\) ? \(G v=\) Some True
            ( \(B\) ) ? \(G v=\) Some False
            by fastforce
    hence cnf (encode-state-variable t(index ? vs v) (?G v))
                            \(=\{\{\) literal-formula-to-literal (encode-state-variable \(t(\) index ?vs v) (?G v))
\}\}
            unfolding encode-state-variable-def
        by (cases, force+)
    \} note \(n b=\) this
    have \(c n f{ }^{?} \Phi_{G}=\bigcup(\{\operatorname{cnf}(\) encode-state-variable \(t(\) index ?vs \(v)(? G v))\)
        \(\mid v . v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right) \wedge ? G v \neq\) None \(\left.\}\right)\)
        unfolding cnf-of-encode-goal-state-set-i
        by blast
    also have \(\ldots=\bigcup\left(\left(\lambda v\right.\right.\). cnf \(\left(\right.\) encode-state-variable \(t(\) index ?vs \(\left.\left.v)\left(\left((\Pi)_{G}\right) v\right)\right)\right)\)
        ' \(\left\{v \mid v . v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right) \wedge\left((\Pi)_{G}\right) v \neq\right.\) None \(\left.\}\right)\)
        using setcompr-eq-image[of
            \(\lambda v . c n f\left(\right.\) encode-state-variable \(t\left(\right.\) index ? ?vs v) \(\left.\left(\left((\Pi)_{G}\right) v\right)\right)\)
            \(\lambda v . v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right) \wedge\left((\Pi)_{G}\right) v \neq\) None \(]\)
        by presburger
    also have \(\ldots=\bigcup((\lambda v\). \(\{\{\) literal-formula-to-literal
        (encode-state-variable \(t(\) index ?vs \(v)(? G v))\}\})\)
        - \{v.v \(\operatorname{set}\left((\Pi)_{\mathcal{V}}\right) \wedge\left((\Pi)_{G}\right) v \neq\) None \(\left.\}\right)\)
        using \(n b\)
        by \(\operatorname{simp}\)
    finally show ?thesis
    unfolding \(n b\)
    by auto
qed
```

- This lemma essentially states that the cnf for the cnf formula for the encoding has a clause for each variable whose state is defined in the goal state with the corresponding literal.

```
lemma cnf-of-encode-goal-state-set:
    fixes \(\Pi:: ~ ' a ~ s t r i p s-p r o b l e m ~\)
    assumes is-valid-problem-strips \(\Pi\)
        and \(v \in \operatorname{dom}\left((\Pi)_{G}\right)\)
    shows \(\left((\Pi)_{G}\right) v=\) Some True \(\longrightarrow\left(\exists!C . C \in \operatorname{cnf}\left(\left(\Phi_{G} \Pi\right) t\right)\right.\)
        \(\left.\wedge C=\left\{(\text { State } t(\text { index }(\text { strips-problem.variables-of } \Pi) v))^{+}\right\}\right)\)
    and \(\left((\Pi)_{G}\right) v=\) Some False \(\longrightarrow\left(\exists!C . C \in \operatorname{cnf}\left(\left(\Phi_{G} \Pi\right) t\right)\right.\)
        \(\left.\wedge C=\left\{(\text { State } t(\text { index }(\text { strips-problem.variables-of } \Pi) v))^{-1}\right\}\right)\)
proof -
    let ?vs \(=\) strips-problem.variables-of \(\Pi\)
        and ? \(G=(\Pi)_{G}\)
        and \(? \Phi_{G}=\left(\Phi_{G} \Pi\right) t\)
    have \(n b_{1}: c n f ? \Phi_{G}=\bigcup\{\) cnf (encode-state-variable \(t\) (index ?vs \(v\) )
                \((? G v)) \mid v . v \in \operatorname{set}((\Pi) \mathcal{V}) \wedge ? G v \neq\) None \(\}\)
        unfolding cnf-of-encode-goal-state-set-i
        by auto
    have \(n b_{2}: v \in\left\{v . v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right) \wedge ? G v \neq\right.\) None \(\}\)
        using is-valid-problem-dom-of-goal-state-is assms(1, 2)
        by auto
    have \(n b_{3}\) : cnf (encode-state-variable \(t\) (index (strips-problem.variables-of \(\Pi\) ) \(v\) )
\(\left.\left(\left((\Pi)_{G}\right) v\right)\right)\)
    \(\subseteq(\bigcup\{\operatorname{cnf}\) (encode-state-variable \(t\) (index ?vs v)
        \((? G v)) \mid v . v \in \operatorname{set}((\Pi) \mathcal{V}) \wedge ? G v \neq\) None \(\})\)
        using \(U N\)-upper [OF \(n b_{2}\), of \(\lambda v\). cnf (encode-state-variable \(t\) (index ?vs v) (?G
\(v)\) )] \(n b_{2}\)
    by blast
    show \(\left((\Pi)_{G}\right) v=\) Some True \(\longrightarrow\left(\exists!C . C \in \operatorname{cnf}\left(\left(\Phi_{G} \Pi\right) t\right)\right.\)
        \(\left.\wedge C=\left\{(\text { State } t(\text { index }(\text { strips-problem.variables-of } \Pi) v))^{+}\right\}\right)\)
    and \(\left((\Pi)_{G}\right) v=\) Some False \(\longrightarrow\left(\exists!C . C \in \operatorname{cnf}\left(\left(\Phi_{G} \Pi\right) t\right)\right.\)
        \(\left.\wedge C=\left\{(\text { State } t(\text { index }(\text { strips-problem.variables-of } \Pi) v))^{-1}\right\}\right)\)
    using \(n b_{3}\)
    unfolding \(n b_{1}\) encode-state-variable-def
    by auto+
qed
end
```

We omit the proofs that the partial encoding functions produce formulas in CNF form due to their more technical nature. The following sublocale proof confirms that definition ?? encodes a valid problem $\Pi$ into a formula that can be transformed to CNF (is-cnf ( $\Phi \Pi t$ )) and that its CNF has the required form.

### 7.3 Soundness of the Basic SATPlan Algorithm

lemma valuation-models-encoding-cnf-formula-equals:
assumes is-valid-problem-strips $\Pi$
shows $\mathcal{A} \models \Phi \Pi t=c n f$-semantics $\mathcal{A}(\operatorname{cnf}(\Phi \Pi t))$

```
proof -
    let ?\Phi = Ф Пt
    {
        have is-cnf ?\Phi
            using is-cnf-encode-problem[OF assms].
    hence is-nnf ?\Phi
            using is-nnf-cnf
            by blast
    }
    thus ?thesis
    using cnf-semantics[of ?\Phi \mathcal{A]}
    by blast
qed
corollary valuation-models-encoding-cnf-formula-equals-corollary:
    assumes is-valid-problem-strips \Pi
    shows }\mathcal{A}\models(\Phi\Pit
        =(\forallC\in\operatorname{cnf}(\Phi\Pit).\existsL\inC.lit-semantics \mathcal{A L})
    using valuation-models-encoding-cnf-formula-equals[OF assms]
    unfolding cnf-semantics-def clause-semantics-def encode-problem-def
    by presburger
- A couple of technical lemmas about decode-plan.
lemma decode-plan-length:
    assumes }\pi=\mp@subsup{\Phi}{}{-1}\Pi\nu
    shows length }\pi=
    using assms
    unfolding decode-plan-def SAT-Plan-Base.decode-plan-def
    by simp
lemma decode-plan'-set-is[simp]:
    set (decode-plan' П \mathcal{A k)}
    ={(strips-problem.operators-of \Pi)!(index (strips-problem.operators-of \Pi) op)
        | op.op \in set (strips-problem.operators-of \Pi)
        \wedge \mathcal { A } ( \text { Operator k (index (strips-problem.operators-of П) op)) \}}
proof -
    let ?ops = strips-problem.operators-of \Pi
    let ?f = \lambdaop. Operator k (index ?ops op)
    let ?vs = map ?f ?ops
    {
        have set (filter \mathcal{A ?vs) = set (map ?f (filter (\mathcal{A}\circ?f) ?ops))})
            unfolding filter-map[of \mathcal{A \lambdaop. Operator k (index ?ops op) ?ops]..}
    hence set (filter \mathcal{A ?vs) = (\lambdaop. Operator k (index ?ops op))'}
                {op\in set ?ops. \mathcal{A (Operator k (index ?ops op)) }}
            unfolding set-map set-filter
        by simp
    }
    have set (decode-plan' \Pi\mathcal{A k})=(\lambdav. case v of Operator k i=> ?ops ! i)
```

' ( lop. Operator $k$ (index ?ops op))' $\{$ op $\in$ set ?ops. $\mathcal{A}$ (Operator $k$ (index ?ops op ) ) \}
unfolding decode-plan'-def set-map Let-def
by auto
also have $\ldots=($ ( $o p$. case Operator $k$ (index ?ops op) of Operator $k i \Rightarrow$ ?ops ! i)
' $\{$ op $\in$ set ?ops. $\mathcal{A}($ Operator $k$ (index ?ops op $))\}$
unfolding image-comp comp-apply
by argo
also have $\ldots=(\lambda o p$. ?ops ! (index ?ops op $)$ )
' $\{$ op $\in$ set ?ops. $\mathcal{A}$ (Operator $k$ (index ?ops op)) \}
by force
finally show?thesis
by blast
qed
lemma decode-plan-set-is[simp]:
set $\left(\Phi^{-1} \Pi \mathcal{A} t\right)=(\bigcup k \in\{. .<t\} .\{$ decode-plan' $\Pi \mathcal{A} k\})$
unfolding decode-plan-def SAT-Plan-Base.decode-plan-def set-map
using atLeast-upt
by blast
lemma decode-plan-step-element-then- $i$ :
assumes $k<t$
shows set $\left(\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\right)$
$=\{$ (strips-problem.operators-of $\Pi)!($ index (strips-problem.operators-of $\Pi)$ op)
$\mid$ op. op $\in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right) \wedge \mathcal{A}$ (Operator $k$ (index (strips-problem.operators-of $\Pi$ )
$o p)$ ) $\}$
proof -
have $\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k=$ decode-plan' $\Pi \mathcal{A} k$
unfolding decode-plan-def SAT-Plan-Base.decode-plan-def
using assms
by $\operatorname{simp}$
thus ?thesis
by force
qed

- Show that each operator op in the $k$-th parallel operator in a decoded parallel plan is contained within the problem's operator set and the valuation is true for the corresponding SATPlan variable.
lemma decode-plan-step-element-then:
fixes $\Pi$ ::'a strips-problem
assumes $k<t$
and $o p \in \operatorname{set}\left(\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\right)$
shows $o p \in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)$
and $\mathcal{A}$ (Operator $k$ (index (strips-problem.operators-of $\Pi$ ) op))
proof -
let ?ops $=$ strips-problem.operators-of $\Pi$
let ?Ops $=\{$ ?ops! (index ?ops op)

```
    |op.op }\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{})\wedge\mathcal{A}(\mathrm{ Operator }k\mathrm{ (index ?ops op)) }
    have op }\in\mathrm{ ?Ops
    using assms(2)
    unfolding decode-plan-step-element-then-i[OF assms(1)] assms
    by blast
    moreover have op \in set ((\Pi)
    and \mathcal{A (Operator k (index ?ops op))}
    using calculation
    by fastforce+
    ultimately show op \in set ((\Pi)
    and \mathcal{A (Operator k (index ?ops op))}
    by blast+
qed
- Show that the \(k\)-th parallel operators of the decoded plan are distinct lists (i.e. do not contain duplicates).
lemma decode-plan-step-distinct:
    assumes k<t
    shows distinct (( }\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t)!k
proof -
    let ?ops = strips-problem.operators-of \Pi
        and ? }\mp@subsup{\pi}{k}{}=(\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t)!
    let ?f = \lambdaop. Operator k (index ?ops op)
    and ?g=\lambdav.case v of Operator - k=> ?ops! k
    let ?vs = map ?f (remdups ?ops)
    have n\mp@subsup{b}{1}{}: ? }\mp@subsup{\pi}{k}{}=\mathrm{ decode-plan' П A k
        unfolding decode-plan-def SAT-Plan-Base.decode-plan-def
    using assms
    by fastforce
{
    have distinct (remdups ?ops)
        by blast
    moreover have inj-on ?f (set (remdups ?ops))
            unfolding inj-on-def
        by fastforce
    ultimately have distinct ?vs
        using distinct-map
        by blast
    } note n\mp@subsup{b}{2}{}=this
    {
        have inj-on ?g (set ?vs)
            unfolding inj-on-def
            by fastforce
        hence distinct (map ?g ?vs)
            using distinct-map nb 
            by blast
}
thus ?thesis
    using distinct-map-filter[of ?g ?vs \mathcal{A}
```

```
    unfolding nb l decode-plan'-def Let-def
    by argo
qed
lemma decode-state-at-valid-variable:
    fixes \Pi :: 'a strips-problem
    assumes ( }\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}k)v\not=Non
    shows v\in set ((\Pi)\mathcal{V})
proof -
    let ?vs = strips-problem.variables-of \Pi
    let ?f = \lambdav. (v,\mathcal{A}(\mathrm{ State k (index ?vs v)))}
    {
```



```
        by force
        also have ... = (\lambdav. fst (v,\mathcal{A}(\mathrm{ State k (index ?vs v))))' set ?vs}
            by blast
        finally have fst 'set (map ?f ?vs) = set ?vs
        by auto
    }
    moreover have }\negv\not\infst'set (map ?f ?vs
        using map-of-eq-None-iff[of map ?f ?vs v] assms
        unfolding decode-state-at-def SAT-Plan-Base.decode-state-at-def
        by meson
    ultimately show ?thesis
        by fastforce
qed
- Show that there exists an equivalence between a model \(\mathcal{A}\) of the (CNF of the) encoded problem and the state at step \(k\) decoded from the encoded problem.
lemma decode-state-at-encoding-variables-equals-some-of-valuation-if:
fixes \(\Pi\) :: 'a strips-problem
assumes is-valid-problem-strips \(\Pi\)
and \(\mathcal{A} \models \Phi \Pi t\)
and \(k \leq t\)
and \(v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\)
shows \(\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A} k\right) v\)
\(=\) Some \((\mathcal{A}(\) State \(k(\) index (strips-problem.variables-of \(\Pi) v)))\)
proof -
let ?vs \(=\) strips-problem.variables-of \(\Pi\)
let ?l \(=\operatorname{map}(\lambda x .(x, \mathcal{A}(\) State \(k(\) index ?vs \(x))))\) ?vs
have set ? vs \(\neq\{ \}\)
using assms(4)
by fastforce
then have map-of ?l \(v=\operatorname{Some}(\mathcal{A}(\) State \(k(\) index ?vs \(v)))\)
using map-of-from-function-graph-is-some-if[of ?vs \(v\)
\(\lambda v . \mathcal{A}(\) State \(k\) (index ?vs \(v)\) )] \(\operatorname{assms}(4)\)
by fastforce
thus ?thesis
unfolding decode-state-at-def SAT-Plan-Base.decode-state-at-def
```

```
    by meson
qed
lemma decode-state-at-dom:
    assumes is-valid-problem-strips \Pi
    shows dom}(\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}k)=\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{V}}{}
proof-
    let ?s = \Phi }\mp@subsup{S}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}
        and ?vs = strips-problem.variables-of \Pi
    have dom?s = fst'set (map ( }\lambdav.(v,\mathcal{A}(\mathrm{ State k (index ?vs v)))) ?vs)
        unfolding decode-state-at-def SAT-Plan-Base.decode-state-at-def
        using dom-map-of-conv-image-fst[of (map (\lambdav. (v,\mathcal{A (State k (index ?vs v))))}
?vs)]
        by meson
    also have ... = fst' ( }\lambdav.(v,\mathcal{A}(\mathrm{ State k (index ?vs v))))' set ((П)v)
        using set-map[of (\lambdav. (v,\mathcal{A (State k (index ?vs v)))) ?vs]}]
        by simp
    also have .. = (fst \circ (\lambdav. (v,\mathcal{A}(\mathrm{ State k (index ?vs v)))))' set ((П)v)})=\mp@code{v})
        using image-comp[of fst (\lambdav. (v,\mathcal{A}(\mathrm{ State k (index ?vs v))))]}]
        by presburger
    finally show ?thesis
        by force
qed
lemma decode-state-at-initial-state:
    assumes is-valid-problem-strips \Pi
        and }\mathcal{A}=\Phi\Pi
    shows}(\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}0)=(\Pi\mp@subsup{)}{I}{
proof -
    let ?I = (\Pi) }\mp@subsup{I}{}{\prime
    let ?s = 和 }\mp@subsup{}{}{-1}\Pi\mathcal{A}
    let ?vs = strips-problem.variables-of \Pi
    let ?\Phi=\Phi Пt
    let ? }\mp@subsup{\Phi}{I}{}=\mp@subsup{\Phi}{I}{}
    {
        have is-cnf ? }\mp@subsup{\Phi}{I}{}\mathrm{ and cnf ? }\mp@subsup{\Phi}{I}{}\subseteqcnf ?
            subgoal
                using is-cnf-encode-initial-state[OF assms(1)]
                by simp
            subgoal
                using cnf-of-encode-problem-structure(1)
                unfolding encode-initial-state-def encode-problem-def
                by blast
            done
        then have cnf-semantics \mathcal{A (cnf ? }\mp@subsup{\Phi}{I}{})
            using cnf-semantics-monotonous-in-cnf-subsets-if is-cnf-encode-problem[OF
assms(1)]
                assms(2)
```

by blast
hence $\forall C \in \operatorname{cnf}$ ? $\Phi_{I}$. clause-semantics $\mathcal{A} C$
unfolding cnf-semantics-def encode-initial-state-def
by blast
$\}$ note $n b_{1}=t h i s$
\{
\{
fix $v$
assume $v$-in-dom- $i: v \in$ dom?I
moreover \{
have $v$-in-variable-set: $v \in$ set $\left((\Pi)_{\mathcal{V}}\right)$
using is-valid-problem-strips-initial-of-dom assms(1) v-in-dom-i by auto
hence $\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A} 0\right) v=\operatorname{Some}(\mathcal{A}($ State $0($ index ?vs $v)))$
using decode-state-at-encoding-variables-equals-some-of-valuation-if[OF
$\operatorname{assms}(1,2)-v$-in-variable-set]
by fast
$\}$ note $n b_{2}=$ this
consider ( $v$-initially-true) ?I $v=$ Some True
| (v-initially-false) ?I $v=$ Some False
using $v$-in-dom-i
by fastforce
hence ?I $v=$ ?s $v$
proof (cases)
case $v$-initially-true
then obtain $C$
where $C \in \operatorname{cnf} ? \Phi_{I}$
and $c$-is: $C=\left\{(\text { State } 0(\text { index ? vs } v))^{+}\right\}$
using cnf-of-encode-initial-state-set $v$-in-dom-i assms(1)
by fastforce
hence $\mathcal{A}($ State $0($ index ?vs $v))=$ True
using $n b_{1}$
unfolding clause-semantics-def
by fastforce
thus ?thesis
using $n b_{2}$ v-initially-true
by presburger
next
case $v$-initially-false
then obtain $C$
where $C \in \operatorname{cnf} ? \Phi_{I}$
and $c$-is: $C=\left\{\left(\right.\right.$ State $\left.0(\text { index ?vs v) })^{-1}\right\}$
using cnf-of-encode-initial-state-set assms(1) v-in-dom-i
by fastforce
hence $\mathcal{A}$ (State 0 (index ?vs $v)$ ) $=$ False
using $n b_{1}$
unfolding clause-semantics-def

```
                by fastforce
            thus ?thesis
            using nb 
            by presburger
        qed
    }
    hence ?I }\mp@subsup{\subseteq}{m}{}\mathrm{ ?s
    using map-le-def
    by blast
} moreover {
    {
        fix v
    assume v-in-dom-s:v\indom?s
    then have v-in-set-vs: v\in set ?vs
        using decode-state-at-dom[OF assms(1)]
        by simp
    have v-in-dom-I: v\in dom ?I
        using is-valid-problem-strips-initial-of-dom assms(1) v-in-set-vs
        by auto
    have s-v-is: (程}\mp@subsup{}{}{-1}\Pi\mathcal{A}0)v=Some (\mathcal{A (State 0 (index ?vs v)))
    using decode-state-at-encoding-variables-equals-some-of-valuation-if assms(1,
2)
                v-in-set-vs
        by (metis le0)
    consider (s-v-is-some-true) ?s v}=\mathrm{ Some True
        | (s-v-is-some-false) ?s v = Some False
        using v-in-dom-s
        by fastforce
    hence ?s v = ?I v
        proof (cases)
            case s-v-is-some-true
            then have \mathcal{A}-of-s-v: lit-semantics \mathcal{A ((State 0 (index ?vs v))+})
                using s-v-is
                    by fastforce
                consider (I-v-is-some-true) ?I v = Some True
                    | (I-v-is-some-false) ?I v = Some False
                    using v-in-dom-I
                    by fastforce
        thus ?thesis
            proof (cases)
                case I-v-is-some-true
                then show ?thesis
                    using s-v-is-some-true
                    by argo
                    next
                case I-v-is-some-false
                    then obtain C
                        where C-in-encode-initial-state: }C\in\mathrm{ cnf ? }\mp@subsup{\Phi}{I}{
```

```
                    and C-is:C={(State 0 (index ?vs v))}\mp@subsup{)}{}{-1}
                    using cnf-of-encode-initial-state-set assms(1) v-in-dom-I
                    by fastforce
                    hence lit-semantics \mathcal{A ((State 0 (index ?vs v))}\mp@subsup{)}{}{-1})
                    using n\mp@subsup{b}{1}{}
                    unfolding clause-semantics-def
                    by fast
                    thus ?thesis
                    using \mathcal{A}-of-s-v
                    by fastforce
        qed
    next
        case s-v-is-some-false
        then have \mathcal{A}
            using s-v-is
            by fastforce
        consider (I-v-is-some-true) ?I v = Some True
            | (I-v-is-some-false) ?I v = Some False
            using v-in-dom-I
            by fastforce
        thus ?thesis
            proof (cases)
                    case I-v-is-some-true
            then obtain C
                    where C-in-encode-initial-state: }C\incnf ? '\Phi \Phi I
                        and C-is: C={(State 0 (index ?vs v))+}
                    using cnf-of-encode-initial-state-set assms(1) v-in-dom-I
                    by fastforce
            hence lit-semantics \mathcal{A ((State 0 (index ?vs v))+)}
                    using n\mp@subsup{b}{1}{}
                    unfolding clause-semantics-def
                    by fast
            thus ?thesis
                    using \mathcal{A}-of-s-v
                    by fastforce
            next
                case I-v-is-some-false
                    thus ?thesis
                    using s-v-is-some-false
                    by presburger
            qed
    qed
}
hence ?s }\mp@subsup{\subseteq}{m}{}\mathrm{ ?I
    using map-le-def
    by blast
} ultimately show ?thesis
using map-le-antisym
by blast
```

```
qed
lemma decode-state-at-goal-state:
    assumes is-valid-problem-strips \Pi
    and }\mathcal{A}\models\Phi\Pi
    shows (\Pi)}\mp@subsup{)}{G}{}\mp@subsup{\subseteq}{m}{}\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}
proof -
    let ?vs = strips-problem.variables-of \Pi
        and ?G}=(\Pi\mp@subsup{)}{G}{
        and ?}\mp@subsup{G}{}{\prime}=\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}
        and ?\Phi=\Phi \Pit
        and ? }\mp@subsup{\Phi}{G}{}=(\mp@subsup{\Phi}{G}{}\Pi)
    {
        have is-cnf ??\mp@subsup{\Phi}{G}{}\mathrm{ and cnf ? }\mp@subsup{\Phi}{G}{}\subseteqcnf ?\Phi
            subgoal
                using encode-goal-state-is-cnf[OF assms(1)]
                by simp
            subgoal
                using cnf-of-encode-problem-structure(2)
                unfolding encode-goal-state-def encode-problem-def
                by blast
            done
    then have cnf-semantics \mathcal{A (cnf ? }\mp@subsup{\Phi}{G}{})
                using cnf-semantics-monotonous-in-cnf-subsets-if is-cnf-encode-problem[OF
assms(1)]
                assms(2)
            by blast
    hence }\forallC\incnf ?\mp@subsup{\Phi}{G}{}.clause-semantics A C
                unfolding cnf-semantics-def encode-initial-state-def
                by blast
    } note nb }=thi
    {
        fix v
        assume v\in set ((\Pi)\mathcal{V})
        moreover have set ?vs }\not={
            using calculation(1)
            by fastforce
    moreover have ( }\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}t
            =map-of (map (\lambdav. (v,\mathcal{A (State t (index ?vs v)))) ?vs)}
            unfolding decode-state-at-def SAT-Plan-Base.decode-state-at-def
            by metis
            ultimately have ( }\mp@subsup{\Phi}{S}{-1}\Pi\mathcal{A}t)v=\mathrm{ Some ( }\mathcal{A}(\mathrm{ State t (index ?vs v)))
            using map-of-from-function-graph-is-some-if
            by fastforce
    } note n\mp@subsup{b}{2}{}=this
    {
    fix v
```

```
    assume v-in-dom-G:v\indom ?G
    then have v-in-vs:v\in set ?vs
        using is-valid-problem-dom-of-goal-state-is assms(1)
        by auto
    then have decode-state-at-is: ( }\mp@subsup{\Phi}{S}{-1}\Pi\mathcal{A}t)v=Some (\mathcal{A}\mathrm{ (State t (index ?vs
v)))
    using n\mp@subsup{b}{2}{}
    by fastforce
consider (A) ?G v = Some True
    | (B) ?G v = Some False
    using v-in-dom-G
    by fastforce
hence ?G v = ?G'v
    proof (cases)
        case }
        {
            obtain C where C\subseteqcnf ? }\mp@subsup{\Phi}{G}{}\mathrm{ and }C={{(\mathrm{ State t (index ?vs v))+ }}
                using cnf-of-encode-goal-state-set(1)[OF assms(1) v-in-dom-G] A
                by auto
            then have {(State t (index ?vs v))+}}\incnf ?.\mp@subsup{\Phi}{G}{
                by blast
            then have clause-semantics }\mathcal{A}{(\mathrm{ State t (index ?vs v))+}
                    using nb
                    by blast
            then have lit-semantics \mathcal{A }((\mathrm{ State t (index ?vs v))+})
                    unfolding clause-semantics-def
                    by blast
            hence \mathcal{A (State t (index ?vs v)) = True}\=\mp@code{T}
            by force
        }
        thus ?thesis
            using decode-state-at-is A
            by presburger
    next
        case B
        {
            obtain C where C\subseteqcnf ?.\mp@subsup{\Phi}{G}{}\mathrm{ and C={{(State t (index ?vs v))}\mp@subsup{)}{}{-1}}}
                using cnf-of-encode-goal-state-set(2)[OF assms(1) v-in-dom-G] B
                by auto
            then have {(State t (index ?vs v))}\mp@subsup{)}{}{-1}}\incnf ?\mp@subsup{\Phi}{G}{
                by blast
            then have clause-semantics \mathcal{A {(State t (index ?vs v))}\mp@subsup{)}{}{-1}}
                using n\mp@subsup{b}{1}{}
                by blast
            then have lit-semantics \mathcal{A }((\mathrm{ State t (index ?vs v))}\mp@subsup{)}{}{-1})
                unfolding clause-semantics-def
                by blast
            hence \mathcal{A}}(\mathrm{ State t (index ?vs v)) = False
                by simp
```

```
        }
        thus ?thesis
            using decode-state-at-is B
            by presburger
    qed
    }
    thus ?thesis
        using map-le-def
        by blast
qed
- Show that the operator activation implies precondition constraints hold at every time step of the decoded plan.
lemma decode-state-at-preconditions:
assumes is-valid-problem-strips \(\Pi\)
and \(\mathcal{A} \models \Phi \Pi t\)
and \(k<t\)
and \(o p \in \operatorname{set}\left(\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\right)\)
and \(v \in \operatorname{set}\) (precondition-of op)
shows \(\mathcal{A}\) (State \(k\) (index (strips-problem.variables-of П) v))
proof -
let ?ops \(=\) strips-problem.operators-of \(\Pi\)
and ?vs \(=\) strips-problem.variables-of \(\Pi\)
let \(? \Phi=\Phi \Pi t\)
and \(? \Phi_{O}=\) encode-operators \(\Pi t\)
and \({ }^{2} \Phi_{P}=\) encode-all-operator-preconditions \(\Pi\) ? ops \(t\)
\{
have \(\mathcal{A}\) (Operator \(k\) (index ?ops op))
and \(o p \in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)\)
using decode-plan-step-element-then[OF \(\operatorname{assms}(3,4)]\)
by blast+
moreover obtain \(C\)
where clause-is-in-operator-encoding: \(C \in c n f ? \Phi_{P}\)
and \(C=\left\{(\text { Operator } k(\text { index ?ops op }))^{-1}\right.\),
(State \(k\) (index ?vs v)) \(\left.{ }^{+}\right\}\)
using cnf-of-encode-all-operator-preconditions-contains-clause-if [OF assms(1,
3)
calculation(2) assms(5)]
by blast
moreover have clause-semantics- \(\mathcal{A}-\Phi_{P}: \forall C \in\) cnf ? \(\Phi_{P}\). clause-semantics \(\mathcal{A}\)
C
using cnf-semantics-monotonous-in-cnf-subsets-if[OF assms(2)
is-cnf-encode-problem[OF assms(1)]
cnf-of-operator-precondition-encoding-subset-encoding]
unfolding cnf-semantics-def
by blast
ultimately have lit-semantics \(\mathcal{A}(\) Pos (State \(k\) (index ?vs \(v))\) )
unfolding clause-semantics-def
```

```
        by fastforce
}
    thus ?thesis
    unfolding lit-semantics-def
    by fastforce
qed
```

- This lemma shows that for a problem encoding with makespan zero for which a model exists, the goal state encoding must be subset of the initial state encoding. In this case, the state variable encodings for the goal state are included in the initial state encoding.

```
lemma encode-problem-parallel-correct- \(i\) :
    assumes is-valid-problem-strips \(\Pi\)
        and \(\mathcal{A} \models \Phi \Pi 0\)
    shows \(\operatorname{cnf}\left(\left(\Phi_{G} \Pi\right) 0\right) \subseteq \operatorname{cnf}\left(\Phi_{I} \Pi\right)\)
proof -
    let \(? v s=\) strips-problem.variables-of \(\Pi\)
        and \(? I=(\Pi)_{I}\)
        and ? \(G=(\Pi)_{G}\)
        and \(? \Phi_{I}=\Phi_{I} \Pi\)
        and \(? \Phi_{G}=\left(\Phi_{G} \Pi\right) 0\)
        and \(? \Phi=\Phi \Pi 0\)
```

- Show that the model of the encoding is also a model of the partial encodings. have $\mathcal{A}$-models- $\Phi_{I}: \mathcal{A} \models$ ? $\Phi_{I}$ and $\mathcal{A}$-models- $\Phi_{G}: \mathcal{A} \models$ ? $\Phi_{G}$
using assms(2) encode-problem-has-model-then-also-partial-encodings(1, 2)
unfolding encode-problem-def encode-initial-state-def encode-goal-state-def
by blast+
    - Show that every clause in the CNF of the goal state encoding $\Phi_{G}$ is also in the
CNF of the initial state encoding $\Phi_{I}$ thus making it a subset. We can conclude this
from the fact that both $\Phi_{I}$ and $\Phi_{G}$ contain singleton clauses-which must all be
evaluated to true by the given model $\mathcal{A}$-and the similar structure of the clauses
in both partial encodings.
By extension, if we decode the goal state $G$ and the initial state $I$ from a model of
the encoding, $G v=I v$ must hold for variable $v$ in the domain of the goal state.
\{
fix $C^{\prime}$
assume $C^{\prime}-i n-c n f-\Phi_{G}: C^{\prime} \in c n f ? \Phi_{G}$
then obtain $v$
where $v$-in-vs: $v \in$ set ? vs
and $G$-of-v-is-not-None: ? $G v \neq$ None
and $C^{\prime}$-is: $C^{\prime}=\{$ literal-formula-to-literal (encode-state-variable 0 (index
? vs $v$ )
$(? G v))\}$
using cnf-of-encode-goal-state-set-ii[OF assms(1)]
by auto
obtain $C$
where $C$-in-cnf- $\Phi_{I}: C \in c n f ? \Phi_{I}$
and $C$-is: $C=\{$ literal-formula-to-literal (encode-state-variable 0 (index ?vs v) (?I v) ) \}
using cnf-of-encode-initial-state-set-ii[OF assms(1)] v-in-vs
by auto
\{
let $? L=$ literal-formula-to-literal (encode-state-variable 0 (index ?vs v) (?I
v))
have $\{? L\} \in c n f ? \Phi_{I}$
using $C$-in-cnf- $\Phi_{I} C$ - $i s$
by blast
hence lit-semantics $\mathcal{A}$ ? $L$
using model-then-all-singleton-clauses-modelled $[$ OF
is-cnf-encode-initial-state $\left[O F\right.$ assms(1)]- $\mathcal{A}$-models- $\left.\Phi_{I}\right]$
by blast
\} note lit-semantics- $\mathcal{A}-L=$ this
\{
let $? L^{\prime}=$ literal-formula-to-literal (encode-state-variable 0 (index ?vs v) (?G
v))
have $\left\{? L^{\prime}\right\} \in \operatorname{cnf} ? \Phi_{G}$
using $C^{\prime}$-in-cnf- $\Phi_{G} C^{\prime}-i s$
by blast
hence lit-semantics $\mathcal{A}$ ? $L^{\prime}$
using model-then-all-singleton-clauses-modelled $[O F$
encode-goal-state-is-cnf $\left[\right.$ OF assms(1)]- $\mathcal{A}$-models- $\left.\Phi_{G}\right]$
by blast
\} note lit-semantics- $\mathcal{A}-L^{\prime}=$ this
\{
have ?I $v=$ ? $G v$
proof (rule ccontr)
assume contradiction: ?I $v \neq$ ? $G v$
moreover have ?I $v \neq$ None
using $v$-in-vs is-valid-problem-strips-initial-of-dom assms(1)
by auto
ultimately consider (A) ?I $v=$ Some True $\wedge$ ? $G v=$ Some False
| (B) ?I $v=$ Some False $\wedge$ ? $G v=$ Some True
using $G$-of-v-is-not-None
by force
thus False
using lit-semantics- $\mathcal{A}$-L lit-semantics- $\mathcal{A}-L^{\prime}$
unfolding encode-state-variable-def
by (cases, fastforce+)
qed
\}
hence $C^{\prime} \in c n f ? \Phi_{I}$
using $C$-is $C$-in-cnf- $\Phi_{I} C^{\prime}$-is $C^{\prime}$-in-cnf- $\Phi_{G}$
by argo
\}
thus ?thesis


## by blast qed

- Show that the encoding secures that for every parallel operator ops decoded from the plan at every time step $t<$ length $p i$ the following hold:

1. ops is applicable, and
2. the effects of ops are consistent.
lemma encode-problem-parallel-correct-ii:
assumes is-valid-problem-strips $\Pi$
and $\mathcal{A} \models \Phi \Pi t$
and $k<$ length $\left(\Phi^{-1} \Pi \mathcal{A} t\right)$
shows are-all-operators-applicable $\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A} k\right)$
$\left(\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\right)$
and are-all-operator-effects-consistent $\left(\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\right)$
proof -
let ?vs = strips-problem.variables-of $\Pi$
and ?ops $=$ strips-problem.operators-of $\Pi$
and $? \pi=\Phi^{-1} \Pi \mathcal{A} t$
and ? $s=\Phi_{S}{ }^{-1} \Pi \mathcal{A} k$
let $? \Phi=\Phi \Pi t$
and ${ }^{2} \Phi_{E}=$ encode-all-operator-effects $\Pi$ ?ops $t$
have $k$-lt-t: $k<t$
using decode-plan-length assms(3)
by metis
\{
\{
fix $o p v$
assume op-in-kth-of-decoded-plan-set: op $\in \operatorname{set}(? \pi!k)$
and $v$-in-precondition-set: $v \in$ set (precondition-of op)
\{
have $\mathcal{A}$ (Operator $k$ (index ?ops op))
using decode-plan-step-element-then[OF k-lt-t op-in-kth-of-decoded-plan-set]
by blast
hence $\mathcal{A}$ (State $k$ (index ?vs $v$ ))
using decode-state-at-preconditions
OF $\operatorname{assms}(1,2)$ - op-in-kth-of-decoded-plan-set $v$-in-precondition-set]
$k-l t-t$
by blast
\}
moreover have $k \leq t$
using $k$ - $l t$ - $t$
by auto
moreover \{
have $o p \in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)$
using decode-plan-step-element-then[OF k-lt-t op-in-kth-of-decoded-plan-set]
by $\operatorname{simp}$
then have $v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)$
```
            using is-valid-problem-strips-operator-variable-sets(1) assms(1)
            v-in-precondition-set
        by auto
    }
    ultimately have ( }\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}k)v=\mathrm{ Some True
        using decode-state-at-encoding-variables-equals-some-of-valuation-if[OF
assms(1, 2)]
        by presburger
    }
    hence are-all-operators-applicable ?s (?\pi ! k)
        using are-all-operators-applicable-set[of ?s ?\pi !k]
        by blast
} moreover {
    {
        fix }o\mp@subsup{p}{1}{}o\mp@subsup{p}{2}{
        assume op 1-in-k-th-of-decoded-plan:op 
        and op 2-in-k-th-of-decoded-plan:op}\mp@subsup{2}{2}{}\in\operatorname{set}((\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t)!k
    have op (-in-set-ops:op
        and op 2-in-set-ops:op op }\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{}
        and op -active-at-k:\neglit-semantics \mathcal{A ((Operator k (index ?ops op 1 ) )}
        and op,
        subgoal
            using decode-plan-step-element-then[OF k-lt-t op 1-in-k-th-of-decoded-plan]
            by simp
            subgoal
                    using decode-plan-step-element-then[OF k-lt-t op 2-in-k-th-of-decoded-plan]
                    by force
            subgoal
                    using decode-plan-step-element-then[OF k-lt-t op (in-k-th-of-decoded-plan]
                    by simp
            subgoal
                    using decode-plan-step-element-then[OF k-lt-t op2-in-k-th-of-decoded-plan]
                    by simp
            done
        {
            fix v
            assume v-in-add-effects-set-of-op }:v\in\operatorname{set}(add-effects-of op () 
            and v-in-delete-effects-set-of-op 2:v\in set (delete-effects-of op 2)
            let ?C}\mp@subsup{C}{1}{}={(\mathrm{ Operator }k(\mathrm{ index ?ops op 1 ) )}\mp@subsup{)}{}{-1}\mathrm{ ,
                (State (Suc k) (index ?vs v))+}
                and ? }\mp@subsup{C}{2}{}={(\mathrm{ Operator k (index ?ops op 2 ) )}\mp@subsup{)}{}{-1}\mathrm{ ,
                (State (Suc k) (index ?vs v))}\mp@subsup{)}{}{-1}
            have ?C}\mp@subsup{C}{1}{}\incnf?\mp@subsup{\Phi}{E}{}\mathrm{ and ? }\mp@subsup{C}{2}{}\incnf? ?\mp@subsup{\Phi}{E}{
                subgoal
                    using cnf-of-operator-effect-encoding-contains-add-effect-clause-if[OF
                    assms(1) k-lt-t op -in-set-ops v-in-add-effects-set-of-op 1]
                    by blast
                    subgoal
```

using cnf-of-operator-effect-encoding-contains-delete-effect-clause-if [OF $\operatorname{assms}(1) k$-lt-t op $p_{2}$-in-set-ops $v$-in-delete-effects-set-of-op ${ }_{2}$ ]
by blast
done
then have ? $C_{1} \in c n f ? \Phi$ and ? $C_{2} \in c n f ? \Phi$
using cnf-of-encode-all-operator-effects-subset-cnf-of-encode-problem by blast+
then have $C_{1}$-true: clause-semantics $\mathcal{A}$ ? $C_{1}$ and $C_{2}$-true: clause-semantics $\mathcal{A}$ ? $C_{2}$
using valuation-models-encoding-cnf-formula-equals[OF assms(1)] assms(2) unfolding cnf-semantics-def by blast+
have lit-semantics $\mathcal{A}\left(\left(\right.\right.$ State $($ Suc $\left.k)(\text { index ?vs v) })^{+}\right)$
and lit-semantics $\mathcal{A}\left((\text { State }(k+1)(\text { index ?vs } v))^{-1}\right)$
subgoal
using op $p_{1}$-active-at-k $C_{1}$-true
unfolding clause-semantics-def
by blast
subgoal
using op ${ }_{2}$-active-at-k $C_{2}$-true
unfolding clause-semantics-def
by fastforce
done
hence False
by auto
\} moreover \{
fix $v$
assume $v$-in-delete-effects-set-of-op $: v \in$ set (delete-effects-of op $p_{1}$ )
and $v$-in-add-effects-set-of-op $p_{2}: v \in$ set (add-effects-of op ${ }_{2}$ )
let ? $C_{1}=\left\{\left(\text { Operator } k\left(\text { index ?ops op } p_{1}\right)\right)^{-1},\left(\right.\right.$ State $($ Suc $\left.k)(\text { index ?vs v) })^{-1}\right\}$ and ? $C_{2}=\left\{(\text { Operator } k(\text { index ?ops op } 2))^{-1}\right.$, (State (Suc $\left.k\right)$ (index ?vs
$\left.v))^{+}\right\}$
have ? $C_{1} \in c n f ? \Phi_{E}$ and ${ }^{?} C_{2} \in c n f ? \Phi_{E}$
subgoal
using cnf-of-operator-effect-encoding-contains-delete-effect-clause-if[OF $\operatorname{assms}(1) k$-lt-t op $p_{1}$-in-set-ops $v$-in-delete-effects-set-of-op $p_{1}$ ]
by fastforce
subgoal
using cnf-of-operator-effect-encoding-contains-add-effect-clause-if[OF $\operatorname{assms}(1) k$-lt-t op $p_{2}$-in-set-ops v-in-add-effects-set-of-op $p_{2}$ ]
by $\operatorname{simp}$
done
then have ? $C_{1} \in c n f ? \Phi$ and $? C_{2} \in c n f ? \Phi$
using cnf-of-encode-all-operator-effects-subset-cnf-of-encode-problem
by blast+
then have $C_{1}$-true: clause-semantics $\mathcal{A} ? C_{1}$ and $C_{2}$-true: clause-semantics $\mathcal{A}$ ? $C_{2}$
using valuation-models-encoding-cnf-formula-equals[OF assms(1)] assms(2) unfolding cnf-semantics-def

```
            by blast+
            have lit-semantics \mathcal{A ((State (Suc k) (index ?vs v))}\mp@subsup{)}{}{-1})
            and lit-semantics \mathcal{A ((State (k+1) (index ?vs v))+})
            subgoal
                using op 1-active-at-k C1-true
                    unfolding clause-semantics-def
                    by blast
            subgoal
                using op 2-active-at-k C2-true
                    unfolding clause-semantics-def
                    by fastforce
            done
        hence False
            by simp
        }
        ultimately have set (add-effects-of op (1) \cap set (delete-effects-of op ()}={
            and set (delete-effects-of op () \cap set (add-effects-of op ( ) = {}
            by blast+
    }
    hence are-all-operator-effects-consistent (?\pi!k)
        using are-all-operator-effects-consistent-set[of ?\pi!k]
        by blast
}
ultimately show are-all-operators-applicable ?s (?\pi ! k)
    and are-all-operator-effects-consistent (?\pi!k)
    by blast+
qed
```

- Show that for all operators $o p$ at timestep $k$ of the plan $\Phi^{-1} \Pi \mathcal{A} t$ decoded from the model $\mathcal{A}$, both add effects as well as delete effects will hold in the next timestep Suc $k$.
lemma encode-problem-parallel-correct-iii:
assumes is-valid-problem-strips $\Pi$
and $\mathcal{A} \models \Phi \Pi t$
and $k<$ length $\left(\Phi^{-1} \Pi \mathcal{A} t\right)$
and $o p \in \operatorname{set}\left(\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\right)$
shows $v \in$ set (add-effects-of op)
$\longrightarrow\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A}(\right.$ Suc $\left.k)\right) v=$ Some True
and $v \in$ set (delete-effects-of op)
$\longrightarrow\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A}(\right.$ Suc $\left.k)\right) v=$ Some False
proof -
let ?ops $=$ strips-problem.operators-of $\Pi$
and ?vs $=$ strips-problem.variables-of $\Pi$
let ${ }^{2} \Phi_{F}=$ encode-all-operator-effects $\Pi$ ?ops $t$
and $? A=\left(\bigcup(t, o p) \in\{0 . .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)\right.$.
$\left\{\left\{\left\{(\text { Operator } t(\text { index ?ops op }))^{-1},(\text { State }(\text { Suc } t)(\text { index ?vs } v))^{+}\right\}\right\}\right.$
$\mid v . v \in \operatorname{set}($ add-effects-of op $)\})$
and $? B=\left(\bigcup(t, o p) \in\{0 . .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)\right.$.
$\left\{\left\{\left\{(\text { Operator } t(\text { index ?ops op }))^{-1}\right.\right.\right.$,
(State (Suc t) (index ?vs v) $\left.\left.)^{-1}\right\}\right\}$
$\mid v . v \in$ set (delete-effects-of op)\})
have $k$-lt-t: $k<t$
using decode-plan-length assms(3)
by metis
have op-is-valid: op $\in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)$
using decode-plan-step-element-then[OF $k$-lt-t assms(4)]
by blast
have $k$-op-included: $(k, o p) \in\left(\{0 . .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)\right)$
using $k$-lt-t op-is-valid
by fastforce
thus $v \in$ set (add-effects-of op)
$\longrightarrow\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A}(\right.$ Suc $\left.k)\right) v=$ Some True
and $v \in$ set (delete-effects-of op)
$\longrightarrow\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A}(\right.$ Suc $\left.k)\right) v=$ Some False
proof (auto)
assume $v$-is-add-effect: $v \in$ set (add-effects-of op)
have $\mathcal{A}$ (Operator $k$ (index ?ops op))
using decode-plan-step-element-then[OF k-lt-t assms(4)] by blast
moreover \{
have $\left.\left\{\left\{(\text { Operator } k(\text { index ?ops op }))^{-1} \text {, (State }(\text { Suc } k)(\text { index ? } v s \text { v } v)\right)^{+}\right\}\right\}$
$\in\left\{\left\{\left\{(\text { Operator } k(\text { index ?ops op }))^{-1},(\text { State }(\text { Suc } k)(\text { index ? ?vs } v))^{+}\right\}\right\}\right.$
$\mid v . v \in \operatorname{set}($ add-effects-of op) $\}$
using $v$-is-add-effect
by blast
then have $\left\{\{(\text { Operator } k \text { (index ?ops op }))^{-1}\right.$, (State (Suc $\left.k\right)$ (index ?vs
$\left.\left.v))^{+}\right\}\right\} \in ? A$
using $k$-op-included cnf-of-operator-encoding-structure
UN-iff [of \{\{(Operator $t$ (index ?ops op) $)^{-1}$, (State (Suc t) (index ?vs
$\left.\left.v))^{+}\right\}\right\}$

$$
\left.-\{0 . .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)\right]
$$

by blast
then have $\left\{(\text { Operator } k(\text { index ?ops op }))^{-1},(\text { State }(\text { Suc } k)(\text { index ? } v s ~ v))^{+}\right\}$ $\in \bigcup$ ? $A$
using Union-iff $\left[\right.$ of $\left\{(\text { Operator } k(\text { index ?ops op }))^{-1}\right.$, (State (Suc k) (index ? vs $\left.v))^{+}\right\}$]
by blast
moreover have $\cup ? A \subseteq c n f$ ? $\Phi_{F}$
using cnf-of-encode-all-operator-effects-structure
by blast
ultimately have $\left\{(\text { Operator } k \text { (index ?ops op) })^{-1}\right.$, (State (Suc $k$ ) (index ?vs $\left.v))^{+}\right\} \in c n f ? \Phi_{F}$
using in-mono[of $\bigcup$ ? A cnf ? $\Phi_{F}$ ]
by presburger
\}
ultimately have $\mathcal{A}$ (State (Suc k) (index ?vs v))
using cnf-of-encode-all-operator-effects-subset-cnf-of-encode-problem $\operatorname{assms}(2)[u n f o l d e d$ valuation-models-encoding-cnf-formula-equals-corollary[OF $\operatorname{assms}(1)]]$
unfolding Bex-def
by fastforce
thus $\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A}(\right.$ Suc $\left.k)\right) v=$ Some True
using assms(1) assms(2)
decode-state-at-encoding-variables-equals-some-of-valuation-if
is-valid-problem-strips-operator-variable-sets(2) $k$-lt-t op-is-valid subsetD $v$-is-add-effect
by fastforce
next
assume $v$-is-delete-effect: $v \in$ set (delete-effects-of op)
have $\mathcal{A}$ (Operator $k$ (index ?ops op))
using decode-plan-step-element-then[OF $k$-lt-t assms(4)]
by blast
moreover \{
have $\left\{\left\{(\text { Operator } k(\text { index ?ops op }))^{-1}\right.\right.$, $\left.\left.(\text { State }(\text { Suc } k)(\text { index ?vs } v))^{-1}\right\}\right\}$ $\in\left\{\left\{\left\{(\text { Operator } k(\text { index ?ops op }))^{-1},(\text { State }(\text { Suc } k)(\text { index ?vs v }))^{-1}\right\}\right\}\right.$
$\mid v . v \in \operatorname{set}($ delete-effects-of op)\}
using $v$-is-delete-effect
by blast
then have $\left\{\left\{(\text { Operator } k(\text { index ?ops op }))^{-1}\right.\right.$, (State (Suc $\left.k\right)$ (index ? vs
$\left.\left.v))^{-1}\right\}\right\} \in ? B$
using $k$-op-included cnf-of-encode-all-operator-effects-structure
UN-iff $\left[\right.$ of $\left\{\{(\text { Operator } t \text { (index ?ops op }))^{-1}\right.$, (State (Suc t) (index ?vs
$\left.\left.v))^{+}\right\}\right\}$

$$
\left.-\{0 . .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)\right]
$$

by blast
then have $\left\{(\text { Operator } k(\text { index ?ops op }))^{-1},(\text { State }(\text { Suc } k)(\text { index ?vs } v))^{-1}\right\}$ $\in \bigcup$ ? $B$
using Union-iff $\left[\right.$ of $\{(\text { Operator } k \text { (index ?ops op }))^{-1}$, (State (Suc k) (index ?vs $\left.v))^{-1}\right\}$ ]
by blast
moreover have $\bigcup ? B \subseteq c n f$ ? $\Phi_{F}$
using cnf-of-encode-all-operator-effects-structure Un-upper2[of $\bigcup$ ?B $\bigcup$ ?A] by fast
ultimately have $\{(\text { Operator } k \text { (index ?ops op }))^{-1}$, (State (Suc k) (index ?vs $\left.v))^{-1}\right\} \in c n f ? \Phi_{F}$
using in-mono[of $\bigcup$ ? $B \mathrm{cnf}$ ? $\Phi_{F}$ ]
by presburger
\}
ultimately have $\neg \mathcal{A}($ State $($ Suc $k)($ index ?vs v))

```
        using cnf-of-encode-all-operator-effects-subset-cnf-of-encode-problem
        valuation-models-encoding-cnf-formula-equals-corollary[OF assms(1)] assms(2)
        by fastforce
    moreover have Suc \(k \leq t\)
        using \(k\) - \(l t-t\)
        by fastforce
    moreover have \(v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\)
    using v-is-delete-effect is-valid-problem-strips-operator-variable-sets(3) assms(1)
        op-is-valid
    by auto
    ultimately show \(\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A}(\right.\) Suc \(\left.k)\right) v=\) Some False
        using decode-state-at-encoding-variables-equals-some-of-valuation-if[OF
\(\operatorname{assms}(1,2)]\)
            by auto
    qed
qed
```

- In broad strokes, this lemma shows that the operator frame axioms ensure that state is propagated-i.e. the valuation of a variable does not change inbetween time steps-, if there is no operator active which has an effect on a given variable a: i.e.

$$
\begin{aligned}
& \mathcal{A} \vDash\left(\neg a_{i} \wedge a_{i+1}\right) \longrightarrow \bigvee\left\{o p_{i}, k: o p_{i} \text { has add effect } a\right\} \\
& \mathcal{A} \vDash\left(a_{i} \wedge \neg a_{i+1}\right) \longrightarrow \bigvee\left\{o p_{i}, k: o p_{i} \text { has delete effect } a\right\}
\end{aligned}
$$

Now, if the disjunctions are empty-i.e. if no operator which is activated at time step $k$ has either a positive or negative effect-, we have by simplification

$$
\begin{aligned}
& \mathcal{A} \vDash \neg\left(\neg a_{i} \wedge a_{i+1}\right) \equiv \mathcal{A} \vDash a_{i} \vee \neg a_{i+1} \\
& \mathcal{A} \vDash \neg\left(a_{i} \wedge \neg a_{i+1}\right) \equiv \mathcal{A} \vDash \neg a_{i} \vee a_{i+1}
\end{aligned}
$$

hence

$$
\begin{aligned}
\mathcal{A} & \vDash\left(\neg a_{i} \vee a_{i+1}\right) \wedge\left(a_{i} \vee \neg a_{i+1}\right) \\
\sim \mathcal{A} & \vDash\left\{\left\{\neg a_{i}, a_{i+1}\right\},\left\{a_{i}, \neg a_{i+1}\right\}\right\}
\end{aligned}
$$

The lemma characterizes this simplification. ${ }^{8}$
lemma encode-problem-parallel-correct-iv:
fixes $\Pi:: ~ ' a ~ s t r i p s-p r o b l e m ~$
assumes is-valid-problem-strips $\Pi$
and $\mathcal{A}=\Phi \Pi t$
and $k<t$
and $v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)$
and $\neg\left(\exists\right.$ op $\in \operatorname{set}\left(\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\right)$.
$v \in \operatorname{set}(a d d$-effects-of op) $\vee v \in \operatorname{set}$ (delete-effects-of op))
shows cnf-semantics $\mathcal{A}\left\{\{(\text { State } k \text { (index (strips-problem.variables-of } \Pi \text { ) } v))^{-1}\right.$

[^6], (State (Suc k) (index (strips-problem.variables-of П) v) $\left.\left.)^{+}\right\}\right\}$
and cnf-semantics $\mathcal{A}\left\{\{\text { (State } k \text { (index (strips-problem.variables-of П) } v \text { ) })^{+}\right.$
, (State (Suc k) (index (strips-problem.variables-of $\Pi$ ) $v$ ) $\left.\left.)^{-1}\right\}\right\}$
proof -
let ?vs $=$ strips-problem.variables-of $\Pi$
and ?ops $=$ strips-problem.operators-of $\Pi$
let ? $\Phi=\Phi \Pi t$
and $? \Phi_{F}=$ encode-all-frame-axioms $\Pi t$
and $? \pi_{k}=\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k$
and $? A=\bigcup(k, v) \in(\{0 . .<t\} \times$ set ? $2 s$ s $)$.
$\left\{\left\{\left\{(\text { State } k \text { (index ?vs v) })^{+} \text {, (State (Suc k) (index ?vs v) }\right)^{-1}\right\}\right.$
$\cup\left\{(\text { Operator } k(\text { index ?ops op }))^{+} \mid\right.$op. op $\in$ set ?ops $\wedge v \in$ set (add-effects-of
$o p) ~\}\}\}$
and $? B=\bigcup(k, v) \in(\{0 . .<t\} \times$ set ? $v s)$.
$\left\{\left\{\left\{(\text { State } k(\text { index ? vs v) }))^{-1}\right.\right.\right.$, (State (Suc k) $\left.(\text { index ?vs v) })^{+}\right\}$
$\cup\left\{(\text { Operator } k(\text { index ?ops op }))^{+} \mid\right.$op. op $\in$ set ?ops $\wedge v \in$ set (delete-effects-of
$o p) ~\}\}\}$
and ?C $=\left\{(\text { State } k(\text { index ?vs v }))^{+},\left(\right.\right.$State $($Suc $\left.k)(\text { index ?vs v) })^{-1}\right\}$
$\cup\left\{(\text { Operator } k(\text { index ?ops op }))^{+} \mid\right.$op. op $\in$ set ?ops $\wedge v \in$ set (add-effects-of op) \}
and ? $C^{\prime}=\left\{(\text { State } k(\text { index ? vs } v))^{-1},\left(\right.\right.$ State $($ Suc $\left.k)(\text { index ?vs v) })^{+}\right\}$
$\cup\left\{(\text { Operator } k(\text { index ?ops op }))^{+} \mid\right.$op. op $\in$ set ?ops $\wedge v \in$ set (delete-effects-of $o p)\}$
have $k$ - $v$-included: $(k, v) \in\left(\{. .<t\} \times \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\right)$
using $\operatorname{assms}(3,4)$
by blast
have operator-encoding-subset-encoding: cnf ? $\Phi_{F} \subseteq c n f$ ? $\Phi$
using cnf-of-encode-problem-structure(4)
unfolding encode-problem-def
by fast

- Given the premise that no operator in $\pi_{k}$ exists with add-effect respectively delete effect $v$, we have the following situation for the EPC (effect precondition) sets:
- assuming $o p$ is in set ?ops, either op is in $\pi_{k}$ (then it doesn't have effect on $v$ and therefore is not in either of the sets), or if is not, then $\mathcal{A}$ (Operator $k$ (index ?ops op) $=\perp$ by definition of decode-plan; moreover,
- assuming op is not in set ?ops-this is implicitely encoded as Operator $k$ (length ?ops) and $\mathcal{A}$ (Operator $k$ (length ?ops)) may or may not be true-, then it's not in either of the sets.
. Altogether, we have the situation that the sets only have members Operator $k$ (index ?ops op) with $\mathcal{A}$ (Operator $k$ (index ?ops op)) $=\perp$, hence the clause can be reduced to the state variable literals.
More concretely, the following proof block shows that the following two conditions hold for the operators:

```
op.op}\in{((\mathrm{ Operator k (index ?ops op))}\mp@subsup{)}{}{+}
    |op.op }\in\mathrm{ set ?ops }\wedgev\in\mathrm{ set (add-effects-of op)}
    \longrightarrow \neg \text { lit-semantics } \mathcal { A } \text { op}
```

and

```
\(\forall\) op. op \(\in\left\{\left((\text { Operator } k(\text { index ?ops op }))^{+}\right)\right.\)
    \(\mid\) op. op \(\in\) set ?ops \(\wedge v \in\) set (delete-effects-of op) \(\}\)
    \(\longrightarrow \neg\) lit-semantics \(\mathcal{A}\) op
```

Hence, the operators are irrelevant for cnf-semantics $\mathcal{A}\{C\}$ where $C$ is a clause encoding a positive or negative transition frame axiom for a given variable $v$ of the problem.

```
{
    let ?add = {((Operator k (index ?ops op))+}
        |op.op set ?ops }\wedgev\in\mathrm{ set (add-effects-of op) }
    and ?delete ={((Operator k (index ?ops op))+}
        |op.op set ?ops }\wedgev\in\mathrm{ set (delete-effects-of op) }
    {
        fix op
        assume operator-encoding-in-add: (Operator k (index ?ops op))+}\in?,ad
        hence }\neg\mathrm{ lit-semantics }\mathcal{A}((\mathrm{ Operator }k(\mathrm{ index ?ops op))}\mp@subsup{)}{}{+}
            proof (cases op \in set ? }\mp@subsup{\pi}{k}{}\mathrm{ )
            case True
            then have v\not\in set (add-effects-of op)
                using assms(5)
                by simp
            then have (Operator k (index ?ops op))+}\not\in?\mathrm{ ?add
                by fastforce
            thus ?thesis
                using operator-encoding-in-add
                by blast
            next
                case False
            then show ?thesis
                        proof (cases op \in set ?ops)
                    case True
                {
                            let ?A = { ?ops!index ?ops op |op.
                        op }\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{})\wedge\mathcal{A}(\mathrm{ Operator }k(\mathrm{ index ?ops op))}
                            assume lit-semantics \mathcal{A}}((\mathrm{ Operator k (index ?ops op))+}
                            moreover have operator-active-at-k: \mathcal{A (Operator k (index ?ops op))}
                                    using calculation
                                    by auto
                    moreover have op \in set ((\Pi)
                        using True
                            by force
                            moreover have (?ops ! index ?ops op) \in?A
                                    using calculation(2, 3)
                                    by blast
                    ultimately have op\in set ? }\mp@subsup{\pi}{k}{
                        using decode-plan-step-element-then-i[OF assms(3)]
                        by auto
```

```
            hence False
                    using False
            by blast
        }
        thus ?thesis
            by blast
    next
        case False
        then have op}\not\in{op\in set ?ops.v\in set (add-effects-of op)
        by blast
    moreover have ?add =
        (\lambdaop. (Operator k (index ?ops op))+}
            '{ op }\in\mathrm{ set ?ops.v }\in\mathrm{ set (add-effects-of op) }
        using setcompr-eq-image[of \lambdaop. (Operator k (index ?ops op))}\mp@subsup{)}{}{+
            \lambdaop.op set ?ops }\wedgev\in\mathrm{ set (add-effects-of op)]
        by blast
    ultimately have (Operator k (index ?ops op))}\mp@subsup{)}{}{+}\not\in\mathrm{ ?add
        by force
    thus ?thesis using operator-encoding-in-add
        by blast
    qed
    qed
} moreover {
fix op
assume operator-encoding-in-delete: ((Operator k (index ?ops op))+})\in\mathrm{ ?delete
    hence }\neg\mathrm{ lit-semantics }\mathcal{A}((\mathrm{ Operator }k(\mathrm{ index ?ops op))+}
    proof (cases op \in set ? }\mp@subsup{\pi}{k}{}\mathrm{ )
        case True
        then have v\not\in set (delete-effects-of op)
            using assms(5)
            by simp
            then have (Operator k (index ?ops op))+ & ?delete
            by fastforce
        thus ?thesis
            using operator-encoding-in-delete
            by blast
next
            case False
            then show ?thesis
            proof (cases op \in set ?ops)
                case True
                {
                    let ?A = { ?ops!index ?ops op |op.
                        op }\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{})\wedge\mathcal{A}(\mathrm{ Operator }k(\mathrm{ index ?ops op))}
                    assume lit-semantics \mathcal{A ((Operator k (index ?ops op))+})
                    moreover have operator-active-at-k: \mathcal{A (Operator k (index ?ops op))}
                        using calculation
                    by auto
```

```
                    moreover have op\in set ((\Pi) () )
                        using True
                        by force
                    moreover have (?ops!index ?ops op) \in?A
                        using calculation(2, 3)
                        by blast
                    ultimately have op set ? }\mp@subsup{\pi}{k}{
                        using decode-plan-step-element-then-i[OF assms(3)]
                    by auto
                    hence False
                    using False
                    by blast
            }
            thus ?thesis
                    by blast
            next
                    case False
                    then have op}\not\in{op\in\mathrm{ set ?ops.v 姕 (delete-effects-of op)}
                    by blast
                    moreover have ?delete =
                    (\lambdaop. (Operator k (index ?ops op))+}
                            '{ op \in set ?ops.v \in set (delete-effects-of op)}
                    using setcompr-eq-image[of \lambdaop. (Operator k (index ?ops op))+
                    \lambdaop.op set ?ops }\wedgev\in\mathrm{ set (delete-effects-of op)]
                    by blast
                    ultimately have (Operator k (index ?ops op))+ }\ddagger\mathrm{ ? delete
                    by force
                    thus ?thesis using operator-encoding-in-delete
                    by blast
            qed
        qed
    }
    ultimately have }\forallop.op\in?add \longrightarrow \neglit-semantics \mathcal{A op
    and}\forallop.op \in?delete \longrightarrow\neglit-semantics \mathcal{A op
    by blast+
} note nb = this
{
    let ?Ops={(Operator k (index ?ops op))+
        | op.op }\in\mathrm{ set ?ops }\wedgev\in\mathrm{ set (add-effects-of op) }
    have ?Ops\subseteq?C
    by blast
    moreover have ?C - ?Ops = {(State k (index ?vs v))+,}(\mathrm{ State (Suc k)
(index ?vs v))}\mp@subsup{)}{}{-1}
    by fast
    moreover have }\forallL\in\mathrm{ ?Ops. ᄀ lit-semantics A L
    using nb(1)
    by blast
```

ultimately have clause-semantics $\mathcal{A}$ ? $C$
$=$ clause-semantics $\mathcal{A}\left\{(\text { State } k \text { (index ?vs v) })^{+}\right.$, (State (Suc k) (index ?vs $\left.v))^{-1}\right\}$
using lit-semantics-reducible-to-subset-if[of ?Ops ?C]
by presburger
\} moreover \{
let ?Ops ${ }^{\prime}=\left\{(\text { Operator } k(\text { index ?ops op }))^{+}\right.$
| op. op $\in$ set ?ops $\wedge v \in$ set (delete-effects-of op) \}
have ? $O p s^{\prime} \subseteq$ ? $C^{\prime}$
by blast
moreover have ? $C^{\prime}-$ ?Ops ${ }^{\prime}=\left\{(\text { State } k(\text { index ?vs } v))^{-1},(\right.$ State $($ Suc $k)$
$\left.(\text { index ?vs v) })^{+}\right\}$
by fast
moreover have $\forall L \in$ ?Ops ${ }^{\prime}$. ᄀ lit-semantics $\mathcal{A} L$
using $n b$ (2)
by blast
ultimately have clause-semantics $\mathcal{A}$ ? $C^{\prime}$
$=$ clause-semantics $\mathcal{A}\left\{(\text { State } k(\text { index ?vs } v))^{-1}\right.$, (State (Suc k) (index ?vs $\left.v))^{+}\right\}$
using lit-semantics-reducible-to-subset-if[of ?Ops' ?C ]
by presburger
\} moreover \{
have cnf-semantics- $\mathcal{A}$ - $\Phi$ :cnf-semantics $\mathcal{A}(c n f$ ? $\Phi)$
using valuation-models-encoding-cnf-formula-equals $[$ OF $\operatorname{assms}(1)] \operatorname{assms}(2)$ by blast
have $k$-v-included: $(k, v) \in(\{. .<t\} \times \operatorname{set}((\Pi) \mathcal{V}))$
using $\operatorname{assms}(3,4)$
by blast
have c-in-un-a: ? $C \in \bigcup ? A$ and $c^{\prime}-i n-u n-b: ? C^{\prime} \in \bigcup ? B$
using $k$ - $v$-included
by force+
then have $? C \in c n f ? \Phi_{F}$ and $? C^{\prime} \in c n f ? \Phi_{F}$
subgoal
using cnf-of-encode-all-frame-axioms-structure UnI1[of ?C $\bigcup$ ?A $\bigcup$ ?B]
c-in-un-a
by metis
subgoal
using cnf-of-encode-all-frame-axioms-structure UnIQ[of ? $\left.C^{\prime} \cup ? B \bigcup ? A\right]$
$c^{\prime}$-in-un-b
by metis
done
then have $\{? C\} \subseteq c n f ? \Phi_{F}$ and $c^{\prime}$-subset-frame-axiom-encoding: $\left\{? C^{\prime}\right\} \subseteq$ cnf ? $\Phi_{F}$
by blast+
then have $\{? C\} \subseteq c n f ? \Phi$ and $\left\{? C^{\prime}\right\} \subseteq c n f ? \Phi$
subgoal
using operator-encoding-subset-encoding
by fast
subgoal
using $c^{\prime}$-subset-frame-axiom-encoding operator-encoding-subset-encoding by fast
done
hence cnf-semantics $\mathcal{A}\{? C\}$ and cnf-semantics $\mathcal{A}\left\{?{ }^{\prime} C^{\prime}\right\}$
using cnf-semantics- $\mathcal{A}-\Phi$ model-for-cnf-is-model-of-all-subsets
by fastforce+
\}
ultimately show cnf-semantics $\mathcal{A}\left\{\left\{(\text { State } k \text { (index ?vs v) })^{-1}\right.\right.$, (State (Suc k) $\left.\left.(\text { index ? vs v) })^{+}\right\}\right\}$
and cnf-semantics $\mathcal{A}\left\{\{(\text { State } k \text { (index ?vs } v))^{+}\right.$, (State (Suc $k$ ) (index ?vs
$\left.\left.v))^{-1}\right\}\right\}$
unfolding cnf-semantics-def
by blast+
qed
lemma encode-problem-parallel-correct-v:
assumes is-valid-problem-strips $\Pi$
and $\mathcal{A} \models \Phi \Pi t$
and $k<$ length $\left(\Phi^{-1} \Pi \mathcal{A} t\right)$
shows $\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A}(\right.$ Suc $\left.k)\right)=$ execute-parallel-operator $\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A} k\right)\left(\left(\Phi^{-1} \Pi\right.\right.$
$\mathcal{A} t)!k)$
proof -
let ?vs = strips-problem.variables-of $\Pi$
and ?ops $=$ strips-problem.operators-of $\Pi$
and $? \pi=\Phi^{-1} \Pi \mathcal{A} t$
and ${ }^{2} s_{k}=\Phi_{S}{ }^{-1} \Pi \mathcal{A} k$
and $? s_{k}{ }^{\prime}=\Phi_{S}{ }^{-1} \Pi \mathcal{A}($ Suc $k)$
let $? t_{k}{ }^{\prime}=$ execute-parallel-operator $? s_{k}(? \pi!k)$ and $? \pi_{k}=? \pi!k$
have $k$-lt- $t: k<t$ and $k$-lte- $t: k \leq t$ and suc- $k$-lte- $t: S u c k \leq t$
using decode-plan-length $[o f$ ? $\pi \Pi \mathcal{A} t] \operatorname{assms}(3)$
by (argo, fastforce + )
then have operator-preconditions-hold:
are-all-operators-applicable ? $s_{k} ? \pi_{k} \wedge$ are-all-operator-effects-consistent $? \pi_{k}$
using encode-problem-parallel-correct-ii[OF $\operatorname{assms}(1,2,3)]$
by blast

- We show the goal in classical fashion by proving that

$$
\begin{aligned}
\Phi_{S}{ }^{-1} & \Pi \mathcal{A}(\text { Suc } k) v \\
& =\text { execute-parallel-operator }\left(\Phi_{S^{-1}} \Pi \mathcal{A} k\right) \\
& \left(\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\right) v
\end{aligned}
$$

-i.e. the state decoded at time $k+1$ is equivalent to the state obtained by executing the parallel operator $\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k$ on the previous state $\Phi_{S}{ }^{-1} \Pi \mathcal{A}$ $k$-for all variables $v$ given $k<t$, a model $\mathcal{A}$, and makespan $t$.
moreover \{

## \{

fix $v$
assume $v$-in-dom- $s_{k}{ }^{\prime}: v \in d o m ? s_{k}{ }^{\prime}$
then have $s_{k}{ }^{\prime}$-not-none: ? $s_{k}{ }^{\prime} v \neq$ None
by blast
hence ? $s_{k}{ }^{\prime} v=? t_{k}{ }^{\prime} v$
proof (cases $\exists o p \in$ set $? \pi_{k} . v \in$ set (add-effects-of op) $\vee v \in$ set
(delete-effects-of op))
case True
then obtain $o p$
where op-in- $\pi_{k}: o p \in$ set $? \pi_{k}$
and $v \in \operatorname{set}$ (add-effects-of op) $\vee v \in \operatorname{set}$ (delete-effects-of op)
by blast
then consider ( $v$-is-add-effect) $v \in \operatorname{set}$ (add-effects-of op)
$\mid(v$-is-delete-effect) $v \in \operatorname{set}$ (delete-effects-of op)
by blast
then show ?thesis
proof (cases)
case $v$-is-add-effect
then have ? $s_{k}{ }^{\prime} v=$ Some True
using encode-problem-parallel-correct-iii(1)[OF $\operatorname{assms}(1,2,3)$
$o p-i n-\pi_{k}$. $v$-is-add-effect
by blast
moreover have are-all-operators-applicable $\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A} k\right)\left(\left(\Phi^{-1} \Pi \mathcal{A}\right.\right.$
$t)!k$
and are-all-operator-effects-consistent $\left(\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\right)$
using operator-preconditions-hold $v$-is-add-effect
by blast+
moreover have $? t_{k}{ }^{\prime} v=$ Some True
using execute-parallel-operator-positive-effect-if[of $\left.\Phi_{S}{ }^{-1} \Pi \mathcal{A} k\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\right] o p-i n-\pi_{k}$
$v$-is-add-effect calculation(2, 3)
by blast
ultimately show ?thesis
by argo
next
case $v$-is-delete-effect
then have ? $s_{k}{ }^{\prime} v=$ Some False
using encode-problem-parallel-correct-iii(2)[OF $\operatorname{assms}(1,2,3)$
$\left.o p-i n-\pi_{k}\right]$
$v$-is-delete-effect
by blast
moreover have are-all-operators-applicable $\left(\Phi_{S}{ }^{-1} \Pi \mathcal{A} k\right)\left(\left(\Phi^{-1} \Pi \mathcal{A}\right.\right.$
$t)!k)$
and are-all-operator-effects-consistent $\left(\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\right)$
using operator-preconditions-hold
by blast+
moreover have $? t_{k}{ }^{\prime} v=$ Some False

```
            using execute-parallel-operator-effect(2) op-in-\pi}\mp@subsup{\pi}{k}{
                v-is-delete-effect calculation(2, 3)
            by fast
            moreover have ? }\mp@subsup{t}{k}{}\mp@subsup{}{}{\prime}v=\mathrm{ Some False
                            by (meson execute-parallel-operator-negative-effect-if op-in-\pi}\mp@subsup{\pi}{k}{}\mathrm{ opera-
tor-preconditions-hold v-is-delete-effect)
            ultimately show ?thesis
            by argo
        qed
next
    case False
    then have ? }\mp@subsup{t}{k}{\prime}\mp@subsup{}{}{\prime}v=?\mp@subsup{s}{k}{}
        using execute-parallel-operator-no-effect-if
        by fastforce
    moreover {
        have v-in-set-vs: v\in set ((\Pi)v)
        using decode-state-at-valid-variable[OF sk}\mp@subsup{}{}{\prime}\mathrm{ '-not-none].
    then have state-propagation-positive:
        cnf-semantics \mathcal{A {{(State k (index ?vs v))}\mp@subsup{)}{}{-1}
            ,(State (Suc k) (index ?vs v))+}}
    and state-propagation-negative:
        cnf-semantics \mathcal{A {{(State k (index ?vs v))+}
            , (State (Suc k) (index ?vs v))}\mp@subsup{)}{}{-1}}
        using encode-problem-parallel-correct-iv[OF assms(1, 2) k-lt-t - False]
        by fastforce+
    consider ( }\mp@subsup{s}{k}{\prime}\mathrm{ 'v-positive) ? }\mp@subsup{s}{k}{\prime}\mp@subsup{}{}{\prime}v=\mathrm{ Some True
        | ( }\mp@subsup{s}{k}{\prime
        using }\mp@subsup{s}{k}{\prime}\mp@subsup{}{}{\prime}\mathrm{ -not-none
        by fastforce
    hence ? s sk}\mp@subsup{}{}{\prime}v=?\mp@subsup{s}{k}{}
        proof (cases)
            case }\mp@subsup{s}{k}{\prime}\mp@subsup{}{}{\prime}v-vositiv
            then have lit-semantics \mathcal{A ((State (Suc k) (index ?vs v))+})
        using decode-state-at-encoding-variables-equals-some-of-valuation-if[OF
                assms(1, 2) suc-k-lte-t v-in-set-vs]
                by fastforce
            then have lit-semantics \mathcal{A ((State k (index ?vs v))+})
                        using state-propagation-negative
                unfolding cnf-semantics-def clause-semantics-def
                by fastforce
            then show ?thesis
        using decode-state-at-encoding-variables-equals-some-of-valuation-if[OF
                assms(1, 2) k-lte-t v-in-set-vs] }\mp@subsup{s}{k}{\prime
                by fastforce
            next
                case }\mp@subsup{s}{k}{}\mp@subsup{}{}{\prime}\mathrm{ -v-negative
            then have }\neg\mathrm{ lit-semantics }\mathcal{A}((\mathrm{ State (Suc k) (index ?vs v))+}
```

```
                    using decode-state-at-encoding-variables-equals-some-of-valuation-if[
                                    OF assms(1, 2) suc-k-lte-t v-in-set-vs]
                    by fastforce
                    then have }\neg\mathrm{ lit-semantics }\mathcal{A}((\mathrm{ State k (index ?vs v))+}
                    using state-propagation-positive
                    unfolding cnf-semantics-def clause-semantics-def
                    by fastforce
                    then show ?thesis
            using decode-state-at-encoding-variables-equals-some-of-valuation-if[OF
                assms(1, 2) k-lte-t v-in-set-vs] }\mp@subsup{s}{k}{\prime}\mp@subsup{}{}{\prime}v-negativ
                    by fastforce
                    qed
            }
            ultimately show ?thesis
                by argo
            qed
    }
hence ? }\mp@subsup{s}{k}{\prime}\mp@subsup{}{}{\prime}\mp@subsup{\subseteq}{m}{}\mp@subsup{}{}{\prime}\mp@subsup{t}{k}{}\mp@subsup{}{}{\prime
    using map-le-def
    by blast
}
moreover {
{
    fix v
    assume v \in dom? ?t '}\mp@subsup{}{}{\prime
    then have }\mp@subsup{t}{k}{\prime}\mathrm{ 'not-none: ?t }\mp@subsup{}{k}{\prime}\mp@subsup{}{}{\prime}v\not=Non
        by blast
    {
        {
            assume contradiction: v\not\in set ((\Pi)\mathcal{V})
            then have ( }\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}k)v=Non
                    using decode-state-at-valid-variable
                    by fastforce
            then obtain op
                    where op-in:op }\in\operatorname{set}((\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t)!k
                        and v-is-or:v\in set (add-effects-of op)
                        v}\in\operatorname{set}\mathrm{ (delete-effects-of op)
                    using execute-parallel-operators-strips-none-if-contraposition[OF
                tk}\mp@subsup{}{}{\prime}\mathrm{ -not-none]
            by blast
            have op-in:op \in set ((\Pi)
                using op-in decode-plan-step-element-then(1) k-lt-t
                by blast
            consider (A) v\in set (add-effects-of op)
                    | (B) v\in set (delete-effects-of op)
                    using v-is-or
                    by blast
            hence False
```

```
            proof (cases)
                    case }
                    then have v\in\operatorname{set}((\Pi)\mathcal{V})
                    using is-valid-problem-strips-operator-variable-sets(2)[OF
                        assms(1)] op-in A
                    by blast
                    thus False
                    using contradiction
                    by blast
                next
                    case }
                    then have v\in set ((\Pi)\mathcal{V})
                    using is-valid-problem-strips-operator-variable-sets(3)[OF
                        assms(1)] op-in B
                    by blast
                    thus False
                    using contradiction
                    by blast
            qed
        }
    }
    hence v-in-set-vs:v\in set ((\Pi)\mathcal{V})
        by blast
    hence ? }\mp@subsup{t}{k}{\prime}\mp@subsup{}{}{\prime}v=? ?\mp@subsup{s}{k}{\prime}\mp@subsup{}{}{\prime}
    proof (cases ( }\exists\mathrm{ op }\in\mathrm{ set ? }\mp@subsup{\pi}{k}{}.v.v\in\mathrm{ set (add-effects-of op) }\veev\in\mathrm{ set (delete-effects-of
op)))
    case True
    then obtain op
        where op-in-set-\pi}\mp@subsup{\pi}{k}{}:op\in\mathrm{ set ? }\mp@subsup{\pi}{k}{
        and v-options: v\in set (add-effects-of op) \veev\in set (delete-effects-of op)
        by blast
    then have op \in set ((\Pi)
        using decode-plan-step-element-then[OF k-lt-t]
        by blast
        then consider (v-is-add-effect) v\in set (add-effects-of op)
            | (v-is-delete-effect) v\in set (delete-effects-of op)
            using v-options
            by blast
    thus ?thesis
        proof (cases)
            case v-is-add-effect
            then have ? }\mp@subsup{t}{k}{}\mp@subsup{}{}{\prime}v=\mathrm{ Some True
                    using execute-parallel-operator-positive-effect-if[OF - op-in-set-\pi}\mp@subsup{\pi}{k}{}
                    operator-preconditions-hold
                    by blast
            moreover have ?s, '}\mp@subsup{}{}{\prime}v=\mathrm{ Some True
                    using encode-problem-parallel-correct-iii(1)[OF assms(1, 2, 3)
op-in-set-\pi}\mp@subsup{\pi}{k}{
                v-is-add-effect
```

```
            by blast
            ultimately show ?thesis
            by argo
        next
            case v-is-delete-effect
            then have ?t t}\mp@subsup{}{}{\prime}v=\mathrm{ Some False
            using execute-parallel-operator-negative-effect-if[OF - -op-in-set-\pi
                operator-preconditions-hold
            by blast
            moreover have ? }\mp@subsup{s}{k}{}\mp@subsup{}{}{\prime}v=\mathrm{ Some False
                using encode-problem-parallel-correct-iii(2)[OF assms(1, 2, 3)
op-in-set- }\mp@subsup{\pi}{k}{}
                v-is-delete-effect
            by blast
            ultimately show ?thesis
                by argo
            qed
    next
    case False
    have state-propagation-positive:
        cnf-semantics \mathcal{A {{(State k (index ?vs v))}\mp@subsup{)}{}{-1}\mathrm{ , (State (Suc k) (index ?vs}
v)\mp@subsup{)}{}{+}}}
    and state-propagation-negative:
    cnf-semantics \mathcal{A {{(State k (index ?vs v))+,(State (Suc k) (index ?vs}
v) )}\mp@subsup{}{}{-1}}
    using encode-problem-parallel-correct-iv[OF assms(1, 2) k-lt-t v-in-set-vs
            False]
    by blast+
    {
    have all-op-in-set-\pi}\mp@subsup{\pi}{k}{}\mathrm{ -have-no-effect:
                            \forallop\in set ? }\mp@subsup{\pi}{k}{}.v|\mathrm{ set (add-effects-of op) }\wedgev\not\in\mathrm{ set (delete-effects-of
op)
        using False
        by blast
    then have ? }\mp@subsup{t}{k}{\prime
    using execute-parallel-operator-no-effect-if[OF all-op-in-set-\pi}\mp@subsup{\pi}{k}{}\mathrm{ -have-no-effect]
        by blast
    } note }\mp@subsup{t}{k}{}\mp@subsup{}{}{\prime}\mathrm{ -equals-s}\mp@subsup{s}{k}{}=thi
    {
    have ? }\mp@subsup{s}{k}{}v\not=Non
        using }\mp@subsup{t}{k}{\prime}\mp@subsup{}{}{\prime}\mathrm{ -not-none }\mp@subsup{t}{k}{\prime}\mp@subsup{}{}{\prime}\mathrm{ -equals-sk
        by argo
    then consider ( }\mp@subsup{s}{k}{}-v\mathrm{ -is-some-true) ? }\mp@subsup{s}{k}{}v=\mathrm{ Some True
        | (sk-v-is-some-false)? ? sk v = Some False
        by fastforce
        }
        then show ?thesis
            proof (cases)
            case }\mp@subsup{s}{k}{}\mathrm{ -v-is-some-true
```

```
                    moreover {
                    have lit-semantics }\mathcal{A}((\mathrm{ State k (index ?vs v))+)
                using decode-state-at-encoding-variables-equals-some-of-valuation-if [OF
                        assms(1, 2) k-lte-t v-in-set-vs] sk-v-is-some-true
                    by simp
                    then have lit-semantics \mathcal{A }((\mathrm{ State (Suc k) (index ?vs v))+})
                        using state-propagation-positive
                    unfolding cnf-semantics-def clause-semantics-def
                    by fastforce
                    then have ? }\mp@subsup{s}{k}{\prime}\mp@subsup{}{}{\prime}v=\mathrm{ Some True
                    using decode-state-at-encoding-variables-equals-some-of-valuation-if[OF
                assms(1, 2) suc-k-lte-t v-in-set-vs]
                    by fastforce
                    }
                    ultimately show ?thesis
                    using t\mp@subsup{t}{k}{}\mp@subsup{}{}{\prime}\mathrm{ -equals-s}\mp@subsup{}{k}{}
                    by simp
                    next
                    case }\mp@subsup{s}{k}{}\mathrm{ -v-is-some-false
                    moreover {
                            have lit-semantics \mathcal{A ((State k (index ?vs v))}\mp@subsup{)}{}{-1})
                    using decode-state-at-encoding-variables-equals-some-of-valuation-if[OF
                        assms(1, 2) k-lte-t v-in-set-vs] sk
                    by simp
                            then have lit-semantics \mathcal{A ((State (Suc k) (index ?vs v))}\mp@subsup{)}{}{-1})
                        using state-propagation-negative
                    unfolding cnf-semantics-def clause-semantics-def
                    by fastforce
                    then have ? }\mp@subsup{s}{k}{\prime}\mp@subsup{}{}{\prime}v=\mathrm{ Some False
                    using decode-state-at-encoding-variables-equals-some-of-valuation-if[OF
                        assms(1, 2) suc-k-lte-t v-in-set-vs]
                            by fastforce
                    }
                    ultimately show ?thesis
                    using }\mp@subsup{t}{k}{\prime}\mp@subsup{}{}{\prime}\mathrm{ -equals-s}\mp@subsup{s}{}{\prime
                    by simp
            qed
        qed
    }
    hence ? }\mp@subsup{t}{k}{\prime}\mp@subsup{}{\prime}{}\mp@subsup{\subseteq}{m}{}\mathrm{ ? ? }\mp@subsup{s}{k}{\prime
    using map-le-def
    by blast
}
ultimately show ?thesis
    using map-le-antisym
    by blast
qed
lemma encode-problem-parallel-correct-vi:
```

```
    assumes is-valid-problem-strips \(\Pi\)
    and \(\mathcal{A} \models \Phi \Pi t\)
    and \(k<\) length (trace-parallel-plan-strips \(\left.\left((\Pi)_{I}\right)\left(\Phi^{-1} \Pi \mathcal{A} t\right)\right)\)
    shows trace-parallel-plan-strips \(\left((\Pi)_{I}\right)\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\)
    \(=\Phi_{S}{ }^{-1} \Pi \mathcal{A} k\)
    using assms
proof -
    let ? \(I=(\Pi)_{I}\)
        and \(? \pi=\Phi^{-1} \Pi \mathcal{A} t\)
    let ? \(\tau=\) trace-parallel-plan-strips ?I ? \(\pi\)
    show ?thesis
        using assms
        proof (induction \(k\) )
            case 0
            hence ? \(\tau\) ! \(0=? I\)
                using trace-parallel-plan-strips-head-is-initial-state
                by blast
            moreover have \(\Phi_{S}{ }^{-1} \Pi \mathcal{A} 0=\) ? \(I\)
                using decode-state-at-initial-state \([O F \operatorname{assms}(1,2)]\)
                by \(\operatorname{simp}\)
            ultimately show ?case
                by \(\operatorname{simp}\)
        next
            case (Suc k)
            let \({ }^{2} \tau_{k}=\) trace-parallel-plan-strips ?I ? \(\pi!k\)
                and \(? s_{k}=\Phi_{S}{ }^{-1} \Pi \mathcal{A} k\)
            have \(k\)-lt-length- \(\tau\)-minus-one: \(k<\) length \(? \tau-1\) and \(k\)-lt-length- \(\tau: k<\) length
? \(\tau\)
            using Suc.prems(3)
            by linarith +
    - Use the induction hypothesis to obtain the proposition for the previous step
\(k\). Then, show that applying the \(k\)-th parallel operator in the plan \(\pi\) on either the
state obtained from the trace or decoded from the model yields the same successor
state.
    \{
    have \({ }^{2} \tau!k=\) execute-parallel-plan ?I (take \(k ? \pi\) )
        using trace-parallel-plan-plan-prefix \(k\)-lt-length- \(\tau\)
        by blast
    hence ? \(\tau_{k}=? s_{k}\)
        using \(\operatorname{Suc} . I H[O F \operatorname{assms}(1,2) k\)-lt-length- \(\tau]\)
        by blast
    \}
    moreover have trace-parallel-plan-strips ?I ? \(\pi\) ! Suc k
        \(=\) execute-parallel-operator \({ }^{?} \tau_{k}(? \pi!k)\)
        using trace-parallel-plan-step-effect-is[OF \(k\)-lt-length- \(\tau\)-minus-one \(]\)
        by blast
    moreover \{
    thm Suc.prems(3)
    have length (trace-parallel-plan-strips ?I ? \(\pi\) ) \(\leq\) length \(? \pi+1\)
```

```
            using length-trace-parallel-plan-strips-lte-length-plan-plus-one
            by blast
            then have k< length ? }
                using Suc.prems(3)
                unfolding Suc-eq-plus1
                by linarith
            hence }\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}(\mathrm{ Suc k)
            = execute-parallel-operator ? s
            using encode-problem-parallel-correct-v[OF assms(1, 2)]
            by simp
        }
        ultimately show ?case
        by argo
    qed
qed
lemma encode-problem-parallel-correct-vii:
    assumes is-valid-problem-strips \Pi
    and \mathcal{A}\models\Phi\Pit
    shows length (map (decode-state-at \Pi \mathcal{A})
        [0..<Suc (length ( }\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t))]
    = length (trace-parallel-plan-strips ((\Pi) I) ( }\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t)
proof -
    let ?I = (\Pi) 
    and }\mp@subsup{}{}{2}\pi=\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}
    let ?\sigma = map (decode-state-at \Pi\mathcal{A})[0..<Suc (length ?\pi)]
    and ?\tau = trace-parallel-plan-strips ?I ?\pi
    let ?l = length ?\tau
    let ?k = ?l - 1
    show ?thesis
        proof (rule ccontr)
            assume length-\sigma-neq-length-\tau: length ?\sigma \not= length ?\tau
            {
            have length ? }\sigma=\mathrm{ length ? }\pi+
                by fastforce
            moreover have length ? }\tau\leqlength ? \pi + 
                using length-trace-parallel-plan-strips-lte-length-plan-plus-one
                by blast
            moreover have length ? }\tau<l=length ? \pi + 
                using length-\sigma-neq-length-\tau calculation
                by linarith
            } note n\mp@subsup{b}{1}{}=this
            {
            have 0 < length ?\tau
                using trace-parallel-plan-strips-not-nil..
            then have length ? }\tau-1<length? 
                using n\mp@subsup{b}{1}{}
                by linarith
            } note n\mp@subsup{b}{2}{}= this
```

```
    {
            obtain k' where length ? }\tau=Suc k
            using less-imp-Suc-add[OF length-trace-parallel-plan-gt-0]
            by blast
            hence ?k < length ?\pi
            using nb 
            by blast
    } note nb 
    {
        have ?\tau !?k = execute-parallel-plan ?I (take ?k ?\pi)
            using trace-parallel-plan-plan-prefix[of ?k]
                length-trace-minus-one-lt-length-trace
            by blast
            thm encode-problem-parallel-correct-vi[OF assms(1, 2)] n\mp@subsup{b}{3}{}
            moreover have ( }\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}?k)=?\tau\mp@code{? ?k
            using encode-problem-parallel-correct-vi[OF assms(1, 2)
                    length-trace-minus-one-lt-length-trace]..
    ultimately have ( }\mp@subsup{\Phi}{S}{-1}\Pi\mathcal{A}?k)=\mathrm{ execute-parallel-plan ?I (take ?k ? }\pi\mathrm{ )
            by argo
    } note nb 
    {
        have are-all-operators-applicable ( }\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}?k)(?\pi!?\mp@code{)
            and are-all-operator-effects-consistent (?\pi !?k)
            using encode-problem-parallel-correct-ii(1, 2)[OF assms(1, 2)]nb 3
            by blast+
            - Unsure why calculation(1, 2) is needed for this proof step. Should just
require the default proof.
            moreover have \negare-all-operators-applicable ( }\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}?k)(?\pi!?k
            and \negare-all-operator-effects-consistent (?\pi!?k)
            using length-trace-parallel-plan-strips-lt-length-plan-plus-one-then[OF nb []
                calculation(1, 2)
            unfolding nb }n\mp@subsup{b}{4}{
            by blast+
            ultimately have False
            by blast
        }
        thus False.
    qed
qed
lemma encode-problem-parallel-correct-x:
    assumes is-valid-problem-strips \Pi
        and }\mathcal{A}\models\Phi\Pi
    shows map (decode-state-at \Pi\mathcal{A})
        [0..<Suc (length ( }\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t))
    = trace-parallel-plan-strips ((\Pi) I) ( }\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t
proof -
    let ?I = (\Pi) I
        and ?}\pi=\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}
```

```
let ? \sigma = map (decode-state-at \Pi \mathcal{A) [0..<Suc (length ? \pi)]}]
    and ?\tau = trace-parallel-plan-strips ?I ?\pi
{
    have length ?\tau = length ?\sigma
        using encode-problem-parallel-correct-vii[OF assms]..
    moreover {
        fix }
        assume k-lt-length-\tau: k< length ?\tau
    then have trace-parallel-plan-strips ((\Pi) I) ( }\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t)!
            = 䅪 }\mp@subsup{}{}{-1}\Pi\mathcal{A}
            using encode-problem-parallel-correct-vi[OF assms]
            by blast
    moreover {
            have length ?\tau \leq length ? }\pi+
                using length-trace-parallel-plan-strips-lte-length-plan-plus-one
                by blast
            then have k< length ? \pi +1
                    using k-lt-length-\tau
                by linarith
            then have k<Suc (length ?\pi) - 0
                by simp
            hence ? }\sigma!k=\mp@subsup{\Phi}{S}{}\mp@subsup{}{}{-1}\Pi\mathcal{A}
                    using nth-map-upt[of k Suc (length ?\pi) 0]
                    by auto
        }
        ultimately have ?\tau ! k=? }\sigma!
            by argo
    }
    ultimately have ?\tau = ?\sigma
        using list-eq-iff-nth-eq[of ?\tau ?\sigma]
        by blast
    }
    thus ?thesis
        by argo
qed
lemma encode-problem-parallel-correct-xi:
    fixes \Pi:: 'a strips-problem
    assumes is-valid-problem-strips \Pi
    and \mathcal{A}\models\Phi\Pit
    and ops \in set ( }\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t
    and op\in set ops
shows op \in set ((\Pi)
proof -
    let ?}\mp@subsup{}{}{?}\pi=\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}
    have length ? }\pi=
    using decode-plan-length
    by force
    moreover obtain k where k< length ? \pi and ops =? }\pi\mathrm{ ! k
```

```
    using in-set-conv-nth[of ops ?\pi] assms(3)
    unfolding calculation
    by blast
    ultimately show ?thesis
    using assms(4) decode-plan-step-element-then(1)
    by force
qed
```

To show soundness, we have to prove the following: given the existence of a model $\mathcal{A}$ of the basic SATPlan encoding $\Phi \Pi t$ for a given valid problem $\Pi$ and hypothesized plan length $t$, the decoded plan $\pi \equiv \Phi^{-1} \Pi \mathcal{A} t$ is a parallel solution for $\Pi$.
We show this theorem by showing equivalence between the execution trace of the decoded plan and the sequence of states

$$
\left.\sigma=\operatorname{map}\left(\lambda k . \Phi_{S}{ }^{-1} \Pi \mathcal{A} k\right)[0 . .<\text { Suc (length ? } \pi)\right]
$$

decoded from the model $\mathcal{A}$. Let

```
\tau trace-parallel-plan-strips I }
```

be the trace of $\pi$. Theorem ?? first establishes the equality $\sigma=\tau$ of the decoded state sequence and the trace of $\pi$. We can then derive that $G$ $\subseteq_{m}$ last $\sigma$ by lemma ??, i.e. the last state reached by plan execution (and moreover the last state decoded from the model), satisfies the goal state $G$ defined by the problem. By lemma ??, we can conclude that $\pi$ is a solution for $I$ and $G$.
Moreover, we show that all operators $o p$ in all parallel operators ops $\in$ set $\pi$ are also contained in $\mathcal{O}$. This is the case because the plan decoding function reverses the encoding function (which only encodes operators in $\mathcal{O}$ ).
By definition ?? this means that $\pi$ is a parallel solution for $\Pi$. Moreover $\pi$ has length $t$ as confirmed by lemma . ${ }^{9}$

```
theorem encode-problem-parallel-sound:
    assumes is-valid-problem-strips \(\Pi\)
        and \(\mathcal{A} \models \Phi \Pi t\)
    shows is-parallel-solution-for-problem \(\Pi\left(\Phi^{-1} \Pi \mathcal{A} t\right)\)
    proof -
        let ?ops \(=\) strips-problem.operators-of \(\Pi\)
            and \(? I=(\Pi)_{I}\)
            and \(? G=(\Pi)_{G}\)
            and \({ }^{2} \pi=\Phi^{-1} \Pi \mathcal{A} t\)
    let \(? \sigma=\operatorname{map}\left(\lambda k . \Phi_{S}{ }^{-1} \Pi \mathcal{A} k\right)[0 . .<\) Suc (length ? \(\pi\) )]
            and \(? \tau=\) trace-parallel-plan-strips \(? I ? \pi\)
        \{
```

[^7]```
    have ?}\sigma=?
            using encode-problem-parallel-correct-x[OF assms].
    moreover {
        have length ? }\pi=
            using decode-plan-length
            by auto
    then have ?G }\mp@subsup{\subseteq}{m}{}\mathrm{ last ? }
            using decode-state-at-goal-state[OF assms]
        by simp
    }
    ultimately have }((\Pi\mp@subsup{)}{G}{})\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan ((П)
    using execute-parallel-plan-reaches-goal-iff-goal-is-last-element-of-trace
    by auto
    }
    moreover have }\forallops\in set ?\pi. \forallop \in set ops. op \in set ((\Pi)\mathcal{O}
    using encode-problem-parallel-correct-xi[OF assms(1, 2)]
    by auto
    ultimately show ?thesis
    unfolding is-parallel-solution-for-problem-def
    unfolding list-all-iff ListMem-iff operators-of-def STRIPS-Representation.operators-of-def
    by fastforce
qed
value stop
```


### 7.4 Completeness

definition empty-valuation :: sat-plan-variable valuation $\left(\mathcal{A}_{0}\right)$
where empty-valuation $\equiv(\lambda-$. False $)$
abbreviation valuation-for-state
:: 'variable list
$\Rightarrow$ 'variable strips-state
$\Rightarrow$ nat
$\Rightarrow$ 'variable
$\Rightarrow$ sat-plan-variable valuation
$\Rightarrow$ sat-plan-variable valuation
where valuation-for-state vs skv
$\equiv \mathcal{A}($ State $k($ index vs $v):=(s v=$ Some True $))$

- Since the trace may be shorter than the plan length even though the last trace element subsumes the goal state - namely in case plan execution is impossible due to violation of the execution condition but the reached state serendipitously subsumes the goal state -, we also have to repeat the valuation for all time steps $k^{\prime} \in\{$ length $\tau$..length $\pi+1\}$ for all $v \in \mathcal{V}\left(\right.$ see $\left.\mathcal{A}_{2}\right)$.
definition valuation-for-state-variables
:: 'variable strips-problem
$\Rightarrow$ 'variable strips-operator list list
$\Rightarrow$ 'variable strips-state list

```
\(\Rightarrow\) sat-plan-variable valuation
where valuation-for-state-variables \(\Pi \pi \tau \equiv\) let
    \(t^{\prime}=\) length \(\tau\)
    \(; \tau_{\Omega}=\tau!\left(t^{\prime}-1\right)\)
    ; vs \(=\) variables-of \(\Pi\)
    \(; V_{1}=\left\{\right.\) State \(k(\) index vs \(v) \mid k v . k \in\left\{0 . .<t^{\prime}\right\} \wedge v \in\) set \(\left.v s\right\}\)
    \(; V_{2}=\left\{\right.\) State \(k(\) index vs \(v) \mid k v . k \in\left\{t^{\prime} . .(\right.\) length \(\left.\pi+1)\right\} \wedge v \in\) set vs \(\}\)
    ; \(\mathcal{A}_{1}=\) fold \(r\)
        \((\lambda(k, v) \mathcal{A}\). valuation-for-state (variables-of \(\Pi)(\tau!k) k v \mathcal{A})\)
        (List.product \(\left[0 . .<t^{\prime}\right]\) vs)
        \(\mathcal{A}_{0}\)
    ; \(\mathcal{A}_{2}=\) fold \(r\)
        \((\lambda(k, v) \mathcal{A}\). valuation-for-state (variables-of \(\left.\Pi) \tau_{\Omega} k v \mathcal{A}\right)\)
        (List.product \(\left[t^{\prime} . .<\right.\) length \(\left.\pi+2\right]\) vs)
        \(\mathcal{A}_{0}\)
    in override-on (override-on \(\mathcal{A}_{0} \mathcal{A}_{1} V_{1}\) ) \(\mathcal{A}_{2} V_{2}\)
```

- The valuation is left to yield false for the potentially remaining $k^{\prime} \in\{$ length $\tau$..length $\pi+1\}$ since no more operators are executed after the trace ends anyway. The definition of $\mathcal{A}_{0}$ as the valuation that is false for every argument ensures this implicitely.
definition valuation-for-operator-variables
:: 'variable strips-problem
$\Rightarrow$ 'variable strips-operator list list
$\Rightarrow$ 'variable strips-state list
$\Rightarrow$ sat-plan-variable valuation
where valuation-for-operator-variables $\Pi \pi \tau \equiv$ let
ops $=$ operators-of $\Pi$
$; O p=\{$ Operator $k($ index ops op $) \mid k$ op. $k \in\{0 . .<$ length $\tau-1\} \wedge o p \in$ set ops $\}$
in override-on
$\mathcal{A}_{0}$
(foldr
$(\lambda(k$, op $) \mathcal{A}$. $\mathcal{A}($ Operator $k($ index ops op $):=$ True $))$
(concat $(\operatorname{map}(\lambda k . \operatorname{map}($ Pair $k)(\pi!k))[0 . .<$ length $\tau-1]))$ $\mathcal{A}_{0}$ )
Op
The completeness proof requires that we show that the SATPlan encoding $\Phi \Pi t$ of a problem $\Pi$ has a model $\mathcal{A}$ in case a solution $\pi$ with length $t$ exists. Since a plan corresponds to a state trace $\tau \equiv$ trace-parallel-plan-strips $I \pi$ with

$$
\tau!k=\text { execute-parallel-plan I (take } k \pi)
$$

for all $k<$ length $\tau$ we can construct a valuation $\mathcal{A}_{V}$ modeling the state sequence in $\tau$ by letting
$\mathcal{A}($ State $k($ index vs $v):=(s v=$ Some True $))$
or all $v \in \mathcal{V}$ where $s \equiv \tau!k .{ }^{10}$
Similarly to $\mathcal{A}_{V}$, we obtain an operator valuation $\mathcal{A}_{O}$ by defining
$\mathcal{A}($ Operator $k($ index ops op $):=$ True $)$
for all operators $o p \in \mathcal{O}$ s.t. op $\in \operatorname{set}(\pi!k)$ for all $k<$ length $\tau-1$.
The overall valuation for the plan execution $\mathcal{A}$ can now be constructed by combining the state variable valuation $\mathcal{A}_{V}$ and operator valuation $\mathcal{A}_{O}$.

```
definition valuation-for-plan
    :: 'variable strips-problem
    \(\Rightarrow\) 'variable strips-operator list list
    \(\Rightarrow\) sat-plan-variable valuation
    where valuation-for-plan \(\Pi \pi \equiv\) let
            vs \(=\) variables-of \(\Pi\)
        ; ops \(=\) operators- of \(\Pi\)
        ; \(\tau=\) trace-parallel-plan-strips (initial-of \(\Pi\) ) \(\pi\)
        ; \(t=\) length \(\pi\)
        ; \(t^{\prime}=\) length \(\tau\)
        ; \(\mathcal{A}_{V}=\) valuation-for-state-variables \(\Pi \pi \tau\)
        ; \(\mathcal{A}_{O}=\) valuation-for-operator-variables \(\Pi \pi \tau\)
        ; \(V=\{\) State \(k(\) index vs \(v)\)
        \(\mid k v . k \in\{0 . .<t+1\} \wedge v \in\) set \(v s\}\)
        ; Op \(=\{\) Operator \(k\) (index ops op)
        \(\mid k\) op. \(k \in\{0 . .<t\} \wedge o p \in\) set ops \(\}\)
        in override-on (override-on \(\mathcal{A}_{0} \mathcal{A}_{V} V\) ) \(\mathcal{A}_{O}\) Op
```

- Show that in case of an encoding with makespan zero, it suffices to show that a given model satisfies the initial state and goal state encodings.
lemma model-of-encode-problem-makespan-zero-iff:

$$
\mathcal{A} \models \Phi \Pi 0 \longleftrightarrow \mathcal{A} \models \Phi_{I} \Pi \wedge\left(\Phi_{G} \Pi\right) 0
$$

## proof -

have encode-operators $\Pi 0=\neg \perp \wedge \neg \perp$
unfolding encode-operators-def encode-all-operator-effects-def
encode-all-operator-preconditions-def
by $\operatorname{simp}$
moreover have encode-all-frame-axioms $\Pi 0=\neg \perp$
unfolding encode-all-frame-axioms-def
by $\operatorname{simp}$
ultimately show ?thesis
unfolding encode-problem-def SAT-Plan-Base.encode-problem-def encode-initial-state-def encode-goal-state-def
by $\operatorname{simp}$
qed

[^8]```
lemma empty-valution-is-False[simp]: \(\mathcal{A}_{0} v=\) False
    unfolding empty-valuation-def..
lemma model-initial-state-set-valuations:
    assumes is-valid-problem-strips \(\Pi\)
    shows set (map \(\left(\lambda v\right.\). case \(\left((\Pi)_{I}\right) v\) of Some \(b\)
            \(\Rightarrow \mathcal{A}_{0}(\) State 0 (index (strips-problem.variables-of \(\Pi\) ) \(\left.v):=b\right)\)
        | \(-\Rightarrow \mathcal{A}_{0}\) )
        (strips-problem.variables-of П))
    \(=\left\{\mathcal{A}_{0}(\right.\) State 0 (index (strips-problem.variables-of \(\left.\Pi) v\right):=\) the \(\left.\left(\left((\Pi)_{I}\right) v\right)\right)\)
        \(\left.\mid v . v \in \operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\right\}\)
proof -
    let ? \(I=(\Pi)_{I}\)
    and ?vs \(=\) strips-problem.variables-of \(\Pi\)
    let \(? f=\lambda v\). case \(\left((\Pi)_{I}\right) v\) of Some \(b\)
    \(\Rightarrow \mathcal{A}_{0}\) (State 0 (index ?vs v) \(\left.:=b\right) \mid-\Rightarrow \mathcal{A}_{0}\)
    and ? \(g=\lambda v . \mathcal{A}_{0}(\) State \(0(\) index ? vs \(v):=\) the \((? I v))\)
    let ? \(\mathcal{A s}=\) map ?f ?vs
    have \(n b_{1}\) : dom ? \(I=\operatorname{set}\left((\Pi)_{\mathcal{V}}\right)\)
        using is-valid-problem-strips-initial-of-dom assms
        by fastforce
    \(\{\)
        \{
            fix \(v\)
            assume \(v \in d o m\) ? \(I\)
            hence ?f \(v=\) ? \(g v\)
                    using \(n b_{1}\)
            by fastforce
    \}
    hence ?f' set \(\left((\Pi)_{\mathcal{V}}\right)=? g\) ' set \(\left((\Pi)_{\mathcal{V}}\right)\)
        using \(n b_{1}\)
        by force
    \}
    then have set? \(\mathcal{A} s=? g ' \operatorname{set}((\Pi) \mathcal{V})\)
        unfolding set-map
        by \(\operatorname{simp}\)
    thus ?thesis
    by blast
qed
lemma valuation-of-state-variable-implies-lit-semantics-if:
    assumes \(v \in \operatorname{dom} S\)
    and \(\mathcal{A}(\) State \(k(\) index vs \(v))=\) the \((S v)\)
    shows lit-semantics \(\mathcal{A}\) (literal-formula-to-literal (encode-state-variable \(k\) (index
vs \(v)(S v))\) )
proof -
    let \(? L=\) literal-formula-to-literal (encode-state-variable \(k(\) index vs v) (Sv))
```

```
consider (True) Sv=Some True
    | (False) \(S v=\) Some False
    using assms(1)
    by fastforce
    thus ?thesis
    unfolding encode-state-variable-def
    using assms(2)
    by (cases, force + )
qed
lemma foldr-fun-upd:
    assumes inj-on \(f\) (set \(x s\) )
        and \(x \in\) set \(x s\)
    shows foldr \((\lambda x \mathcal{A}\). \(\mathcal{A}(f x:=g x))\) xs \(\mathcal{A}(f x)=g x\)
    using assms
proof (induction xs)
    case (Cons a xs)
    then show ?case
    proof (cases xs \(=[]\) )
        case True
        then have \(x=a\)
            using Cons.prems(2)
            by \(\operatorname{simp}\)
        thus ?thesis
            by \(\operatorname{simp}\)
    next
            case False
            thus ?thesis
                proof (cases \(a=x\) )
                next
                case False
                \{
                    from False
                    have \(x \in\) set \(x s\)
                            using Cons.prems(2)
                            by \(\operatorname{simp}\)
                            moreover have \(\operatorname{inj}\)-on \(f\) (set xs)
                            using Cons.prems(1)
                            by fastforce
                            ultimately have \((\) foldr \((\lambda x \mathcal{A} . \mathcal{A}(f x:=g x)) x s \mathcal{A})(f x)=g x\)
                            using Cons.IH
                            by blast
                \} moreover \{
                    - Follows from modus tollens on the definition of inj-on.
                    have \(f a \neq f x\)
                            using Cons.prems False
                            by force
                                    moreover have foldr \((\lambda x \mathcal{A}\). \(\mathcal{A}(f x:=g x))(a \# x s) \mathcal{A}\)
```



```
                    by simp
                    ultimately have foldr ( }\lambdax\mathcal{A}.\mathcal{A}(fx:=gx))(a#xs)\mathcal{A}(fx
                        = (foldr (\lambdax \mathcal{A. A (fx := g x)) xs \mathcal{A ) (fx)}}\mathbf{(f)}
                    unfolding fun-upd-def
                    by presburger
            } ultimately show ?thesis
                    by argo
        qed simp
    qed
qed fastforce
lemma foldr-fun-no-upd:
    assumes inj-on f (set xs)
        and y\not\inf' set xs
    shows foldr (\lambdax\mathcal{A}.\mathcal{A}(fx:=gx)) xs \mathcal{A }y=\mathcal{A}y
    using assms
proof (induction xs)
    case (Cons a xs)
    {
        have inj-on f (set xs) and y\not\inf' set xs
            using Cons.prems
            by (fastforce, simp)
    hence foldr (\lambdax\mathcal{A}.\mathcal{A}(fx:=gx)) xs \mathcal{A}y=\mathcal{A}y
            using Cons.IH
            by blast
    }
    moreover {
        have fa\not=y
            using Cons.prems(2)
            by auto
        moreover have foldr ( }\lambdax\mathcal{A}.\mathcal{A}(fx:=gx))(a#xs)\mathcal{A
            =(foldr (\lambdax\mathcal{A. A (f x := g x)) xs \mathcal{A})(f a := g a)}
            by simp
        ultimately have foldr (\lambdax\mathcal{A}.\mathcal{A}(fx:=gx))(a#xs)\mathcal{A}y
            =(foldr (\lambdax\mathcal{A}.\mathcal{A}(fx:=gx)) xs \mathcal{A})y
            unfolding fun-upd-def
            by presburger
    }
    ultimately show ?case
        by argo
qed simp
```

- We only use the part of the characterization of $\mathcal{A}$ which pertains to the state variables here.
lemma encode-problem-parallel-complete- $i$ :
fixes $\Pi:: ' a$ strips-problem
assumes is-valid-problem-strips $\Pi$
and $(\Pi)_{G} \subseteq_{m}$ execute-parallel-plan $\left((\Pi)_{I}\right) \pi$

```
\forallvk.k<length (trace-parallel-plan-strips ((\Pi) I) \pi)
    \longrightarrow ( \mathcal { A } \text { (State k (index (strips-problem.variables-of П) v))}
        \longleftrightarrow (trace-parallel-plan-strips ((\Pi\mp@subsup{)}{I}{})\pi!k)v=Some True)
        \wedge (\neg\mathcal{A (State k (index (strips-problem.variables-of \Pi) v))}
        \longleftrightarrow ( ( \text { trace-parallel-plan-strips } ( ( \Pi ) _ { I } ) \pi ! k ) v \neq ~ S o m e ~ T r u e ) ) ~
    shows \mathcal{A}\models\mp@subsup{\Phi}{I}{}\Pi
proof -
    let ?vs = strips-problem.variables-of \Pi
        and ?I = (\Pi) I
        and ?G = (\Pi) G
        and ? }\mp@subsup{\Phi}{I}{}=\mp@subsup{\Phi}{I}{}
    let ? }\tau=\mathrm{ trace-parallel-plan-strips ?I }
    {
        fix }
        assume C\incnf ? }\mp@subsup{\Phi}{I}{
        then obtain v
            where v-in-set-vs: v\in set ?vs
            and C-is:C={ literal-formula-to-literal (encode-state-variable 0 (index ?vs
v) (?I v)) }
            using cnf-of-encode-initial-state-set-ii[OF assms(1)]
            by auto
        {
            have 0< length ?\tau
            using trace-parallel-plan-strips-not-nil
            by blast
            then have \mathcal{A (State 0 (index (strips-problem.variables-of \Pi) v))}
                \longleftrightarrow trace-parallel-plan-strips ((\Pi\mp@subsup{)}{I}{})\pi!0)v=Some True
            and}\neg\mathcal{A}(\mathrm{ State 0 (index (strips-problem.variables-of П) v))
                \longleftrightarrow ( ( \text { trace-parallel-plan-strips } ( ( \Pi ) _ { I } ) \pi ! 0 ) v \neq \text { Some True)}
            using assms(3)
            by (presburger +)
    } note nb=this
    {
            let ?L = literal-formula-to-literal (encode-state-variable 0 (index ?vs v) (?I
v))
            have \tau-0-is: ? \tau ! 0 = ?I
                using trace-parallel-plan-strips-head-is-initial-state
                by blast
    have v-in-dom-I:v\indom ?I
            using is-valid-problem-strips-initial-of-dom assms(1) v-in-set-vs
            by fastforce
    then consider (I-v-is-Some-True) ?I v = Some True
            | (I-v-is-Some-False) ?I v = Some False
            by fastforce
    hence lit-semantics \mathcal{A ?L}
            unfolding encode-state-variable-def
            using assms(3) \tau-0-is nb
            by (cases, force+)
    }
```

```
    hence clause-semantics \mathcal{A C}
        unfolding clause-semantics-def C-is
        by blast
    }
    thus ?thesis
    using is-cnf-encode-initial-state[OF assms(1)] is-nnf-cnf cnf-semantics
    unfolding cnf-semantics-def
    by blast
qed
```

— Plans may terminate early (i.e. by reaching a state satisfying the goal state before reaching the time point corresponding to the plan length). We therefore have to show the goal by splitting cases on whether the plan successfully terminated early. If not, we can just derive the goal from the assumptions pertaining to $\mathcal{A}$ Otherwise, we have to first show that the goal was reached (albeit early) and that our valuation $\mathcal{A}$ reflects the termination of plan execution after the time point at which the goal was reached.

```
lemma encode-problem-parallel-complete-ii:
    fixes \(\Pi\) ::' a strips-problem
    assumes is-valid-problem-strips \(\Pi\)
    and \((\Pi)_{G} \subseteq_{m}\) execute-parallel-plan \(\left((\Pi)_{I}\right) \pi\)
    and \(\forall v k . k<\) length (trace-parallel-plan-strips \(\left.\left((\Pi)_{I}\right) \pi\right)\)
        \(\longrightarrow(\mathcal{A}\) (State \(k\) (index (strips-problem.variables-of \(\Pi) v))\)
        \(\longleftrightarrow\) (trace-parallel-plan-strips \(\left.\left((\Pi)_{I}\right) \pi!k\right) v=\) Some True)
    and \(\forall v l . l \geq\) length (trace-parallel-plan-strips \(\left.\left((\Pi)_{I}\right) \pi\right) \wedge l<\) length \(\pi+1\)
        \(\longrightarrow \mathcal{A}\) (State l (index (strips-problem.variables-of \(\Pi\) ) \(v)\) )
        \(=\mathcal{A}\left(\right.\) State \(\left(\right.\) length \(\left(\right.\) trace-parallel-plan-strips \(\left.\left.\left((\Pi)_{I}\right) \pi\right)-1\right)\)
            (index (strips-problem.variables-of П) v))
    shows \(\mathcal{A} \models\left(\Phi_{G} \Pi\right)(\) length \(\pi)\)
proof -
    let ?vs = strips-problem.variables-of \(\Pi\)
        and \(? I=(\Pi)_{I}\)
        and ? \(G=(\Pi)_{G}\)
        and \(? \Phi_{I}=\Phi_{I} \Pi\)
        and \(? t=\) length \(\pi\)
        and \(? \Phi_{G}=\left(\Phi_{G} \Pi\right)\) (length \(\left.\pi\right)\)
    let ? \(\tau=\) trace-parallel-plan-strips ?I \(\pi\)
    let \(? t^{\prime}=\) length \(? \tau\)
    \{
        fix \(v\)
        assume \(G\)-of-v-is-not-None: ? \(G v \neq\) None
        have ? \(G \subseteq_{m}\) last ? \(\tau\)
        using execute-parallel-plan-reaches-goal-iff-goal-is-last-element-of-trace assms(2)
            by blast
    also have \(\ldots=? \tau!\left(? t^{\prime}-1\right)\)
        using last-conv-nth[OF trace-parallel-plan-strips-not-nil].
    finally have ? \(G \subseteq_{m} ? \tau!\left(? t^{\prime}-1\right)\)
        by argo
    hence \(\left(? \tau!\left(? t^{\prime}-1\right)\right) v=? G v\)
```

using $G$-of-v-is-not-None
unfolding map-le-def
by force
$\}$ note $n b_{1}=$ this

- Discriminate on whether the trace has full length or not and show that the model valuation of the state variables always correspond to the (defined) goal state values.
\{
fix $v$
assume $G$-of-v-is-not-None: ? $G v \neq$ None
hence $\mathcal{A}$ (State ?t (index ?vs $v)) \longleftrightarrow$ ?G $v=$ Some True
proof $\left(\right.$ cases $\left.? t^{\prime}=? t+1\right)$
case True
moreover have ?t $<$ ? $t^{\prime}$
using calculation
by fastforce
moreover have $\mathcal{A}($ State ?t $($ index ? vs $v)) \longleftrightarrow($ ? $\tau$ ! ?t $) v=$ Some True using assms(3) calculation(2)
by blast
ultimately show ?thesis
using $n b_{1}$ [OF G-of-v-is-not-None]
by force
next
case False
\{
have $? t^{\prime}<? t+1$
using length-trace-parallel-plan-strips-lte-length-plan-plus-one False le-neq-implies-less
by blast
moreover have $\mathcal{A}($ State ?t $($ index ?vs $v))=\mathcal{A}\left(\right.$ State $\left(? t^{\prime}-1\right)($ index
?vs $v)$ )
using assms(4) calculation
by $\operatorname{simp}$
moreover have ? $t^{\prime}-1<? t^{\prime}$
using trace-parallel-plan-strips-not-nil length-greater-0-conv[of ? $\tau]$ less-diff-conv2[of 1 ? $t^{\prime}$ ? $\left.t^{\prime}\right]$
by force
moreover have $\mathcal{A}\left(\right.$ State $\left(? t^{\prime}-1\right)($ index ? vs $\left.v)\right) \longleftrightarrow\left(? \tau!\left(? t^{\prime}-1\right)\right) v$ = Some True
using assms(3) calculation(3)
by blast
ultimately have $\mathcal{A}($ State ?t $($ index ?vs $v)) \longleftrightarrow\left(? \tau!\left(? t^{\prime}-1\right)\right) v=$
Some True
by blast
\}
thus ?thesis
using $n b_{1}[O F$ G-of-v-is-not-None]
by presburger

```
    qed
    } note nb 
    {
        fix C
    assume C-in-cnf-of-\Phi}\mp@subsup{\Phi}{G}{}:C\incnf ?.\mp@subsup{\Phi}{G}{
    moreover obtain v
        where v\in set ?vs
            and G-of-v-is-not-None: ?G v}\not=\mathrm{ None
        and C-is:C = { literal-formula-to-literal (encode-state-variable ?t (index ?vs
v)
            (?G v)) }
        using cnf-of-encode-goal-state-set-ii[OF assms(1)] calculation
        by auto
    consider (G-of-v-is-Some-True) ?G v = Some True
        | (G-of-v-is-Some-False) ?G v = Some False
        using G-of-v-is-not-None
        by fastforce
    then have clause-semantics }\mathcal{A}
        using n\mp@subsup{b}{2}{}C-is
        unfolding clause-semantics-def encode-state-variable-def
        by (cases, force+)
}
thus ?thesis
    using cnf-semantics[OF is-nnf-cnf[OF encode-goal-state-is-cnf[OF assms(1)]]]
    unfolding cnf-semantics-def
    by blast
qed
```

- We are not using the full characterization of $\mathcal{A}$ here since it's not needed.
lemma encode-problem-parallel-complete-iii-a:
fixes $\Pi:: ' a$ strips-problem
assumes is-valid-problem-strips $\Pi$
and $(\Pi)_{G} \subseteq_{m}$ execute-parallel-plan $\left((\Pi)_{I}\right) \pi$
and $C \in \mathrm{cnf}$ (encode-all-operator-preconditions $\Pi$ (strips-problem.operators-of
П) $($ length $\pi)$ )
and $\forall k$ op. $k<$ length (trace-parallel-plan-strips $\left.\left((\Pi)_{I}\right) \pi\right)-1$
$\longrightarrow \mathcal{A}($ Operator $k($ index $($ strips-problem.operators-of $\Pi)$ $o p))=(o p \in$ set
$(\pi!k))$
and $\forall l$ op. $l \geq$ length (trace-parallel-plan-strips $\left.\left((\Pi)_{I}\right) \pi\right)-1 \wedge l<$ length $\pi$ $\longrightarrow \neg \mathcal{A}$ (Operator $l$ (index (strips-problem.operators-of $\Pi$ ) op))
and $\forall v k . k<$ length (trace-parallel-plan-strips $\left((\Pi)_{I}\right) \pi$ )
$\longrightarrow(\mathcal{A}$ (State $k$ (index (strips-problem.variables-of $\Pi$ ) $v))$ $\longleftrightarrow\left(\right.$ trace-parallel-plan-strips $\left.\left((\Pi)_{I}\right) \pi!k\right) v=$ Some True $)$
shows clause-semantics $\mathcal{A} C$
proof -
let ?ops $=$ strips-problem.operators-of $\Pi$
and ?vs $=$ strips-problem.variables-of $\Pi$

```
    and ?t = length \pi
let ?\tau = trace-parallel-plan-strips }((\Pi\mp@subsup{)}{I}{})
obtain k op
    where k-and-op-are: (k,op) \in({0..<?t} }\times\mathrm{ set ((П)
    and C\in(\bigcupv\in set (precondition-of op). {{(Operator k (index ?ops op))}\mp@subsup{)}{}{-1
        , (State k (index ?vs v))+}}}
    using cnf-of-encode-all-operator-preconditions-structure assms(3)
        UN-E[of C]
    by auto
then obtain v
    where v-in-preconditions-of-op: v \in set (precondition-of op)
    and C-is:C={(Operator k (index ?ops op) )}\mp@subsup{)}{}{-1},(\mathrm{ State k (index ?vs v) )}\mp@subsup{)}{}{+}
    by blast
thus ?thesis
    proof (cases k<length ? \tau - 1)
    case k-lt-length-\tau-minus-one: True
    thus ?thesis
        proof (cases op \in set ( }\pi!k)
        case True
        {
            have are-all-operators-applicable (?\tau!k) ( }\pi!k
        using trace-parallel-plan-strips-operator-preconditions k-lt-length-\tau-minus-one
                by blast
            then have (?\tau!k)v=Some True
                using are-all-operators-applicable-set v-in-preconditions-of-op True
                by fast
                hence }\mathcal{A}\mathrm{ (State k (index ?vs v))
                    using assms(6) k-lt-length-\tau-minus-one
            by force
        }
        thus ?thesis
            using C-is
            unfolding clause-semantics-def
            by fastforce
        next
        case False
        then have }\neg\mathcal{A}(\mathrm{ Operator }k\mathrm{ (index ?ops op))
            using assms(4) k-lt-length-\tau-minus-one
            by blast
        thus ?thesis
            using C-is
            unfolding clause-semantics-def
            by fastforce
        qed
    next
    case False
    then have k\geq length ? }\tau-1k<?
        using k-and-op-are
```

```
        by(force, simp)
    then have }\neg\mathcal{A}(\mathrm{ Operator }k\mathrm{ (index ?ops op))
        using assms(5)
        by blast
    thus ?thesis
        unfolding clause-semantics-def
        using C-is
        by fastforce
    qed
qed
```

- We are not using the full characterization of $\mathcal{A}$ here since it's not needed.
lemma encode-problem-parallel-complete-iii-b:
fixes $\Pi::$ 'a strips-problem
assumes is-valid-problem-strips $\Pi$
and $(\Pi)_{G} \subseteq_{m}$ execute-parallel-plan $\left((\Pi)_{I}\right) \pi$
and $C \in \mathrm{cnf}$ (encode-all-operator-effects $\Pi$ (strips-problem.operators-of $\Pi$ )
(length $\pi$ ))
and $\forall k$ op. $k<$ length (trace-parallel-plan-strips $\left.\left((\Pi)_{I}\right) \pi\right)-1$
$\longrightarrow \mathcal{A}$ (Operator $k$ (index (strips-problem.operators-of $\Pi$ ) op $)$ ) $=(o p \in$ set
$(\pi!k))$
and $\forall l$ op. $l \geq$ length (trace-parallel-plan-strips $\left.\left((\Pi)_{I}\right) \pi\right)-1 \wedge l<$ length $\pi$ $\longrightarrow \neg \mathcal{A}$ (Operator l (index (strips-problem.operators-of П) op))
and $\forall v k . k<$ length (trace-parallel-plan-strips $\left((\Pi)_{I}\right) \pi$ )
$\longrightarrow(\mathcal{A}$ (State $k$ (index (strips-problem.variables-of $\Pi$ ) $v)$ )
$\longleftrightarrow\left(\right.$ trace-parallel-plan-strips $\left.\left((\Pi)_{I}\right) \pi!k\right) v=$ Some True $)$
shows clause-semantics $\mathcal{A} C$
proof -
let ?ops $=$ strips-problem.operators-of $\Pi$
and ?vs $=$ strips-problem.variables-of $\Pi$
and $? t=$ length $\pi$
let ? $\tau=$ trace-parallel-plan-strips $\left((\Pi)_{I}\right) \pi$
let $? A=\left(\bigcup(k, o p) \in\{0 . .<? t\} \times \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)\right.$.
$\bigcup v \in \operatorname{set}$ (add-effects-of op).
$\left.\left.\left\{\left\{(\text { Operator } k(\text { index ?ops op }))^{-1} \text {, (State }(\text { Suc } k)(\text { index ? vs v) })\right)^{+}\right\}\right\}\right)$
and $? B=\left(\bigcup(k, o p) \in\{0 . .<? t\} \times \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)\right.$.
$\bigcup v \in$ set (delete-effects-of op).
$\left.\left.\left\{\{(\text { Operator } k \text { (index ?ops op }))^{-1},(\text { State }(\text { Suc } k)(\text { index ?vs } v))^{-1}\right\}\right\}\right)$
consider $(C$-in- $A$ ) $C \in ? A$
$\mid(C-i n-B) C \in$ ? $B$
using Un-iff [of $C$ ? A ?B] cnf-of-encode-all-operator-effects-structure assms(3)
by (metis $C$-in- $A C-i n-B$ )
thus ?thesis
proof (cases)
case $C$-in- $A$
then obtain $k$ op
where $k$-and-op-are: $(k, o p) \in\{0 . .<? t\} \times \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)$
and $C \in(\bigcup v \in \operatorname{set}$ (add-effects-of op).

```
                {{(Operator k (index ?ops op) )}\mp@subsup{)}{}{-1},(\mathrm{ State (Suc k) (index ?vs v))+}}}
        by blast
    then obtain v}\mathrm{ where v-in-add-effects-of-op: v}\in\mathrm{ set (add-effects-of op)
    and C-is:C={(Operator k (index ?ops op))}\mp@subsup{)}{}{-1}\mathrm{ , (State (Suc k) (index ?vs
v)\mp@subsup{)}{}{+}}
    by blast
    thus ?thesis
    proof (cases k<length ?\tau - 1)
        case k-lt-length-\tau-minus-one:True
        thus ?thesis
            proof (cases op \in set ( }\pi!k)
                    case True
                {
                        then have are-all-operators-applicable (? \tau!k) ( }\pi!k
                        and are-all-operator-effects-consistent ( }\pi!k
                using trace-parallel-plan-strips-operator-preconditions k-lt-length-\tau-minus-one
                    by blast+
                            hence execute-parallel-operator (?\tau!k) ( }\pi!k)v=\mathrm{ Some True
                            using execute-parallel-operator-positive-effect-if[
                                OF - True v-in-add-effects-of-op, of ?\tau ! k]
                                by blast
                }
            then have \tau-Suc-k-is-Some-True:(?\tau !Suc k) v = Some True
                            using trace-parallel-plan-step-effect-is[OF k-lt-length-\tau-minus-one]
                    by argo
                    have \mathcal{A (State (Suc k) (index ?vs v))}
                            using assms(6) k-lt-length-\tau-minus-one \tau-Suc-k-is-Some-True
                    by fastforce
                    thus ?thesis
                    using C-is
                    unfolding clause-semantics-def
                    by fastforce
            next
                    case False
                    then have }\neg\mathcal{A}(\mathrm{ Operator k(index ?ops op))
                        using assms(4) k-lt-length-\tau-minus-one
                    by blast
                    thus ?thesis
                    using C-is
                    unfolding clause-semantics-def
                    by force
                qed
    next
        case False
        then have k\geq length ? \tau - 1 and k<?t
            using k-and-op-are
                by auto
            then have }\neg\mathcal{A}(\mathrm{ Operator }k\mathrm{ (index ?ops op))
                using assms(5)
```

```
                by blast
            thus ?thesis
            using C-is
            unfolding clause-semantics-def
            by fastforce
        qed
    next
    - This case is completely symmetrical to the one above.
    case C-in-B
    then obtain kop
        where k-and-op-are: (k,op) \in{0..<?t} }\times\mathrm{ set ((П)
            and C\in(\bigcupv\in set (delete-effects-of op).
            {{(Operator k (index ?ops op))}\mp@subsup{)}{}{-1},(\mathrm{ State (Suc k) (index ?vs v))}\mp@subsup{)}{}{-1}}}
        by blast
    then obtain v where v-in-delete-effects-of-op: v\in set (delete-effects-of op)
        and C-is:C={(Operator k (index ?ops op))}\mp@subsup{)}{}{-1}\mathrm{ , (State (Suc k) (index ?vs
v) )}\mp@subsup{)}{}{-1}
        by blast
    thus ?thesis
        proof (cases k<length ? \tau - 1)
            case k-lt-length-\tau-minus-one: True
            thus ?thesis
                    proof (cases op }\in\operatorname{set}(\pi!k)
                    case True
                {
                        then have are-all-operators-applicable (?\tau!k) ( }\pi!k
                            and are-all-operator-effects-consistent ( }\pi!k
                using trace-parallel-plan-strips-operator-preconditions k-lt-length-\tau-minus-one
                    by blast+
                            hence execute-parallel-operator (?\tau!k) ( }\pi!k)v=\mathrm{ Some False
                            using execute-parallel-operator-negative-effect-if[
                                OF - True v-in-delete-effects-of-op, of ?\tau ! k]
                            by blast
                }
                            then have }\tau\mathrm{ -Suc-k-is-Some-True: (? T!Suc k)v=Some False
                            using trace-parallel-plan-step-effect-is[OF k-lt-length-\tau-minus-one]
                                    by argo
                            have }\neg\mathcal{A}(\mathrm{ State (Suc k) (index ?vs v))
                            using assms(6) k-lt-length-\tau-minus-one \tau-Suc-k-is-Some-True
                    by fastforce
                    thus ?thesis
                        using C-is
                    unfolding clause-semantics-def
                    by fastforce
                next
                    case False
                        then have }\neg\mathcal{A}(\mathrm{ Operator }k\mathrm{ (index ?ops op))
                        using assms(4) k-lt-length-\tau-minus-one
                    by blast
```

```
                    thus ?thesis
                    using C-is
                    unfolding clause-semantics-def
                    by force
                qed
            next
            case False
            then have k\geq length ? }\tau-1\mathrm{ and }k<?
                using k-and-op-are
                by auto
            then have }\neg\mathcal{A}(\mathrm{ Operator k(index ?ops op))
                    using assms(5)
                    by blast
            thus ?thesis
                    using C-is
                    unfolding clause-semantics-def
                    by fastforce
        qed
    qed
qed
lemma encode-problem-parallel-complete-iii:
    fixes \Pi::'a strips-problem
    assumes is-valid-problem-strips \Pi
        and (\Pi)}\mp@subsup{G}{G}{}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan ((П)
        and }\forallk\mathrm{ op. }k<\mathrm{ length (trace-parallel-plan-strips ((П) I) }\pi\mathrm{ ) - 1
            \longrightarrow \mathcal { A } ( \text { Operator k (index (strips-problem.operators-of П) op))} = ( o p \in ~ s e t
(\pi!k))
        and }\foralllop.l\geqlength (trace-parallel-plan-strips ((\Pi) I) \pi) - 1^l<length 
        \longrightarrow \neg \mathcal { A } \text { (Operator l (index (strips-problem.operators-of П) op))}
        and }\forallvk.k<length (trace-parallel-plan-strips ((\Pi\mp@subsup{)}{I}{})\pi
        \longrightarrow ( \mathcal { A } ( \text { State k (index (strips-problem.variables-of П) v))}
                \longleftrightarrow (trace-parallel-plan-strips ((\Pi\mp@subsup{)}{I}{})\pi!k)v=Some True)
    shows \mathcal{A}\models encode-operators \Pi (length \pi)
proof -
    let ?t = length }
        and ?ops = strips-problem.operators-of \Pi
    let ?.\mp@subsup{\Phi}{O}{}= encode-operators \Pi ?t
        and ? }\mp@subsup{\Phi}{P}{}=\mathrm{ encode-all-operator-preconditions П ?ops?t
        and ? }\mp@subsup{\Phi}{E}{}=\mathrm{ encode-all-operator-effects П ?ops ?t
    {
        fix C
        assume C\incnf ? }\mp@subsup{\Phi}{O}{
        then consider (C-in-precondition-encoding) C\incnf ?\Phi}\mp@subsup{\Phi}{P}{
            | (C-in-effect-encoding) C\incnf ?'\Phi
            using cnf-of-operator-encoding-structure
            by blast
        hence clause-semantics \mathcal{A C}
```

```
    proof (cases)
        case C-in-precondition-encoding
        thus ?thesis
            using encode-problem-parallel-complete-iii-a[OF assms(1, 2) - assms(3,
4,5)]
            by blast
    next
        case C-in-effect-encoding
        thus ?thesis
            using encode-problem-parallel-complete-iii-b[OF assms(1, 2)-assms(3, 4,
5)]
            by blast
        qed
}
thus ?thesis
    using encode-operators-is-cnf[OF assms(1)] is-nnf-cnf cnf-semantics
    unfolding cnf-semantics-def
    by blast
qed
lemma encode-problem-parallel-complete-iv-a:
    fixes \Pi :: 'a strips-problem
    assumes STRIPS-Semantics.is-parallel-solution-for-problem \Pi\pi
    and }\forallk\mathrm{ op. }k<\mathrm{ length (trace-parallel-plan-strips ((П) I) }\pi\mathrm{ ) - 1
        \longrightarrow \mathcal { A } \text { (Operator k (index (strips-problem.operators-of П) op))} = ( o p \in ~ s e t
(\pi!k))
    and \forallvk. k< length (trace-parallel-plan-strips ((\Pi\mp@subsup{)}{I}{})\pi)
        \longrightarrow ( \mathcal { A } \text { (State k (index (strips-problem.variables-of П) v))}
        \longleftrightarrow (trace-parallel-plan-strips }((\Pi\mp@subsup{)}{I}{})\pi!k)v=Some True
    and \forallvl.l\geq length (trace-parallel-plan-strips ((\Pi) I) \pi) ^l<length \pi+1
        \longrightarrow \mathcal { A } ( \text { State l (index (strips-problem.variables-of П) v))}
            = \mathcal{A (State}
                (length (trace-parallel-plan-strips ((\Pi\mp@subsup{)}{I}{})\pi) - 1)
                (index (strips-problem.variables-of \Pi) v))
    and}C\in\bigcup(\bigcup(k,v)\in{0..<length \pi} \times set ((\Pi)\mathcal{V})
        {{{(State k (index (strips-problem.variables-of \Pi) v))+
            , (State (Suc k) (index (strips-problem.variables-of \Pi) v))}\mp@subsup{)}{}{-1}
            \cup \{ ( O p e r a t o r ~ k ~ ( i n d e x ~ ( s t r i p s - p r o b l e m . o p e r a t o r s - o f ~ \Pi ) ~ o p ) ) + '
            |op.op\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{})\wedgev\in\operatorname{set}(add-effects-of op) }}})
    shows clause-semantics \mathcal{A C}
proof -
    let ?vs = strips-problem.variables-of \Pi
        and ?ops = strips-problem.operators-of \Pi
        and ?t = length \pi
    let ?\tau = trace-parallel-plan-strips ((\Pi) I) \pi
    let ?A = (U(k,v) \in{0..<?t} }\times\mathrm{ set ?vs.
        {{{(State k (index ?vs v))+, (State (Suc k) (index ?vs v))}\mp@subsup{)}{}{-1}
            \cup \{ ( \text { Operator k (index ?ops op))+ \|op.op 部?ops } \wedge v \in \text { set (add-effects-of}
```

$o p)\}\}\}$ )
$\{$
obtain $C^{\prime}$ where $C^{\prime} \in ? A$ and $C$-in- $C^{\prime}: C \in C^{\prime}$
using Union-iff assms(5)
by auto
then obtain $k v$
where $(k, v) \in\{0 . .<? t\} \times$ set ?vs
and $C^{\prime} \in\left\{\left\{\left\{\left(\right.\right.\right.\right.$ State $k(\text { index ?vs v) })^{+},\left(\right.$State $($Suc $\left.k)(\text { index ?vs v) })^{-1}\right\}$
$\cup\left\{(\text { Operator } k(\text { index ?ops op }))^{+} \mid\right.$op. op $\in$ set ?ops $\wedge v \in$ set (add-effects-of
$o p) ~\}\}\}$
using $U N-E$
by blast
hence $\exists k v$.
$k \in\{0 . .<? t\}$
$\wedge v \in$ set ? vs
$\wedge C=\left\{\left(\right.\right.$ State $\left.k(\text { index ?vs v) })^{+},(\text {State }(\text { Suc } k)(\text { index ?vs } v))^{-1}\right\}$
$\cup\left\{(\text { Operator } k(\text { index ?ops op }))^{+} \mid o p\right.$. op $\in$ set ?ops $\wedge v \in$ set (add-effects-of
$o p)$ \}
using $C$-in- $C^{\prime}$
by blast
\}
then obtain $k v$
where $k$-in: $k \in\{0 . .<? t\}$
and $v$-in-vs: $v \in$ set ?vs
and $C$-is: $C=\left\{(\text { State } k \text { (index ?vs v) })^{+},\left(\right.\right.$State $($Suc $\left.k)(\text { index ?vs v) })^{-1}\right\}$
$\cup\left\{(\text { Operator } k(\text { index ?ops op }))^{+} \mid\right.$op. op $\in$ set ?ops $\wedge v \in$ set (add-effects-of
$o p)\}$
by blast
show ?thesis
proof (cases $k<$ length ? $\tau-1$ )
case $k$-lt-length- $\tau$-minus-one: True
then have $k$-lt-t: $k<? t$
using $k$-in
by force
have all-operators-applicable: are-all-operators-applicable $(? \tau!k)(\pi!k)$
and all-operator-effects-consistent: are-all-operator-effects-consistent $(\pi!k)$
using trace-parallel-plan-strips-operator-preconditions[OF k-lt-length- $\tau$-minus-one] by $\operatorname{simp}+$
then consider $(A) \exists o p \in \operatorname{set}(\pi!k) . v \in \operatorname{set}$ (add-effects-of op)
$\mid(B) \exists o p \in \operatorname{set}(\pi!k) . v \in \operatorname{set}($ delete-effects-of op)
$\mid(C) \forall o p \in \operatorname{set}(\pi!k) . v \notin$ set (add-effects-of op) $\wedge v \notin$ set (delete-effects-of
op)
by blast
thus ?thesis
proof (cases)
case $A$
moreover obtain op

```
    where op-in-\pi}\mp@subsup{\pi}{k}{}:op\in\operatorname{set}(\pi!k
            and v-is-add-effect: v\in set (add-effects-of op)
    using }
    by blast
    moreover {
    have ( }\pi!k)\in\mathrm{ set }
        using k-lt-t
        by simp
    hence op f set ?ops
        using is-parallel-solution-for-problem-operator-set[OF assms(1) -
op-in-\pi}\mp@subsup{\pi}{k}{
            by blast
        }
        ultimately have (Operator k (index ?ops op))+
            { (Operator k (index ?ops op))+ | op. op \in set ?ops }\wedgev\in\mathrm{ set
(add-effects-of op) }
        using v-is-add-effect
        by blast
    then have (Operator k (index ?ops op))+}\in
        using C-is
        by auto
    moreover have \mathcal{A (Operator k (index ?ops op))}
        using assms(2) k-lt-length-\tau-minus-one op-in-\pi
        by blast
    ultimately show ?thesis
        unfolding clause-semantics-def
        by force
    next
    case B
    then obtain op
    where op-in-\pi}\mp@subsup{\pi}{k}{}:op\in\operatorname{set}(\pi!k
            and v-is-delete-effect:v set (delete-effects-of op)..
    then have }\neg(\existsop\in\operatorname{set}(\pi!k).v\in\operatorname{set}(add-effects-of op)
    using all-operator-effects-consistent are-all-operator-effects-consistent-set
        by fast
    then have execute-parallel-operator (?\tau!k) (\pi!k)v
        = Some False
    using execute-parallel-operator-negative-effect-if[OF all-operators-applicable
                all-operator-effects-consistent op-in-\pi
    by blast
    moreover have (?\tau ! Suc k)v= execute-parallel-operator (?\tau!k) (\pi!k)
            using trace-parallel-plan-step-effect-is[OF k-lt-length-\tau-minus-one]
            by simp
            ultimately have }\neg\mathcal{A}(\mathrm{ State (Suc k) (index ?vs v))
            using assms(3) k-lt-length-\tau-minus-one
            by simp
            thus ?thesis
            using C-is
```

```
        unfolding clause-semantics-def
        by simp
    next
        case C
        show ?thesis
        proof (cases (?\tau!k)v=Some True)
            case True
            then have }\mathcal{A}\mathrm{ (State k (index ?vs v))
            using assms(3) k-lt-length-\tau-minus-one
            by force
        thus ?thesis
            using C-is
            unfolding clause-semantics-def
            by fastforce
        next
            case False
            {
                have (?\tau !Suc k) = execute-parallel-operator (?\tau!k) ( }\pi!k
                    using trace-parallel-plan-step-effect-is[OF k-lt-length-\tau-minus-one].
                    then have (?\tau!Suc k)v=(?\tau!k)v
                    using execute-parallel-operator-no-effect-if C
                    by fastforce
                    hence (?\tau!Suc k)v\not= Some True
                    using False
                    by argo
        }
            then have }\neg\mathcal{A}(\mathrm{ State (Suc k) (index ?vs v))
                    using assms(3) k-lt-length-\tau-minus-one
                    by auto
            thus ?thesis
                    using C-is
                    unfolding clause-semantics-def
            by fastforce
        qed
    qed
next
    case k-gte-length-\tau-minus-one: False
    show ?thesis
    proof (cases \mathcal{A (State (length ?\tau - 1) (index ?vs v)))}
        case True
        {
```



```
            proof (cases k= length ?\tau - 1)
                    case False
                    then have length ? }\tau\leqk\mathrm{ and }k<?t+
                    using k-gte-length-\tau-minus-one k-in
                    by fastforce+
                    thus ?thesis
                    using assms(4)
```

```
                by blast
            qed blast
        hence }\mathcal{A}\mathrm{ (State k (index ?vs v))
            using True
            by blast
        }
        thus ?thesis
            using C-is
            unfolding clause-semantics-def
            by simp
next
    case False
    {
            have length ?\tau \leqSuc k and Suc k<?t + 1
            using k-gte-length-\tau-minus-one k-in
            by fastforce+
```



```
(index ?vs v))
            using assms(4)
            by blast
            hence }\neg\mathcal{A}(\mathrm{ State (Suc k) (index ?vs v))
                    using False
                    by blast
            }
            thus ?thesis
                using C-is
                unfolding clause-semantics-def
                by fastforce
            qed
    qed
qed
lemma encode-problem-parallel-complete-iv-b:
fixes \(\Pi\) :: 'a strips-problem
assumes is-parallel-solution-for-problem \(\Pi \pi\)
and \(\forall k\) op. \(k<\) length (trace-parallel-plan-strips \(\left.\left((\Pi)_{I}\right) \pi\right)-1\)
\(\longrightarrow \mathcal{A}\) (Operator \(k\) (index (strips-problem.operators-of \(\Pi\) ) op \()\) ) \(=(o p \in\) set
\((\pi!k))\)
and \(\forall v k . k<\) length (trace-parallel-plan-strips \(\left.\left((\Pi)_{I}\right) \pi\right)\)
\(\longrightarrow(\mathcal{A}\) (State \(k\) (index (strips-problem.variables-of \(\Pi) v))\)
\(\longleftrightarrow\) (trace-parallel-plan-strips \(\left.\left((\Pi)_{I}\right) \pi!k\right) v=\) Some True)
and \(\forall v l . l \geq\) length (trace-parallel-plan-strips \(\left.\left((\Pi)_{I}\right) \pi\right) \wedge l<\) length \(\pi+1\)
\(\longrightarrow \mathcal{A}\) (State l (index (strips-problem.variables-of П) \(v\) ))
\(=\mathcal{A}\) (State
(length (trace-parallel-plan-strips \(\left.\left.\left((\Pi)_{I}\right) \pi\right)-1\right)\) (index (strips-problem.variables-of \(\Pi\) ) \(v\) ))
and \(C \in \bigcup(\bigcup(k, v) \in\{0 . .<\) length \(\pi\} \times \operatorname{set}((\Pi) \mathcal{V})\).
\(\left\{\left\{\{(\text { State } k \text { (index (strips-problem.variables-of } \Pi \text { ) } v))^{-1}\right.\right.\)
```

```
    , (State (Suc k) (index (strips-problem.variables-of \Pi) v))+ }
    \cup \{ ( \text { Operator k (index (strips-problem.operators-of П) op))+}
    |op.op\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{})\wedgev\in\operatorname{set}(\mathrm{ delete-effects-of op) }}})}
    shows clause-semantics \mathcal{A C}
proof -
    let ?vs = strips-problem.variables-of \Pi
        and ?ops = strips-problem.operators-of \Pi
        and ?t = length \pi
    let ?\tau = trace-parallel-plan-strips (initial-of \Pi) }
    let ?A = (\bigcup(k,v) \in{0..<?t} }\times\mathrm{ set ?vs.
        {{{ (State k (index ?vs v))}\mp@subsup{)}{}{-1},(\mathrm{ State (Suc k) (index ?vs v))+ }
            { (Operator k (index ?ops op))+
                |op.op 䪨 ((\Pi\mp@subsup{)}{\mathcal{O}}{})\wedgev\in\operatorname{set (delete-effects-of op) }}})}
    {
    obtain C' where C'}\mp@subsup{C}{}{\prime}\in?A\mathrm{ and C-in-C': C }\in\mp@subsup{C}{}{\prime
        using Union-iff assms(5)
        by auto
    then obtain kv
        where (k,v) \in{0..<?t} }\times\mathrm{ set ?vs
        and C'}\mp@subsup{C}{}{\prime}\in{{{(\mathrm{ State k (index ?vs v))}\mp@subsup{)}{}{-1},(\mathrm{ State (Suc k) (index ?vs v))}\mp@subsup{)}{}{+}
        \cup \{ ( \text { Operator k (index ?ops op))+} \| o p . o p ~ s e t ~ ? o p s ~ \wedge v \in ~ s e t ~ ( d e l e t e - e f f e c t s - o f ~
op) }}}
        using UN-E
        by fastforce
    hence \existskv.
        k\in{0..<?t}
        \wedgev\in set?vs
        \wedgeC={(State k(index ?vs v) )}\mp@subsup{)}{}{-1},(\mathrm{ State (Suc k) (index ?vs v))}\mp@subsup{)}{}{+}
        \cup \{ ( O p e r a t o r ~ k ~ ( i n d e x ~ ? o p s ~ o p ) ) + '
            |op.op \in set ((\Pi)}\mp@subsup{)}{\mathcal{O}}{)}\wedgev\in\operatorname{set}(\mathrm{ delete-effects-of op) }
    using C-in-C'
    by auto
}
then obtain kv
    where k-in:k}\in{0..<?t
        and v-in-vs:v\in set ((\Pi)\mathcal{V})
        and C-is:C={(State k (index ?vs v))}\mp@subsup{)}{}{-1},(\mathrm{ State (Suc k) (index ?vs v))}\mp@subsup{)}{}{+}
            { (Operator k (index ?ops op))+
            |op.op\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{})\wedgev\in\operatorname{set}(\mathrm{ delete-effects-of op) }}
    by auto
    show ?thesis
    proof (cases k<length?\tau - 1)
    case k-lt-length-\tau-minus-one: True
    then have k-lt-t: k< ?t
        using k-in
        by force
```

```
    have all-operators-applicable: are-all-operators-applicable (?\tau!k) (\pi!k)
        and all-operator-effects-consistent: are-all-operator-effects-consistent ( }\pi!k
    using trace-parallel-plan-strips-operator-preconditions[OF k-lt-length-\tau-minus-one]
        by simp+
```



```
        | (B) \existsop 䪨 ( }\pi!k).v\in\operatorname{set (add-effects-of op)
        | (C)\forallop\in set ( }\pi!k).v\not\in\operatorname{set (add-effects-of op)}\wedgev\not\in set (delete-effects-of
op)
            by blast
    thus ?thesis
        proof (cases)
            case }
            moreover obtain op
                where op-in-\mp@subsup{\pi}{k}{}:op\in\operatorname{set}(\pi!k)
                and v-is-delete-effect: v\in set (delete-effects-of op)
            using }
            by blast
            moreover {
                have (\pi!k)\in set \pi
                    using k-lt-t
                by simp
            hence op \in set ?ops
                    using is-parallel-solution-for-problem-operator-set[OF assms(1) -
op-in-\pi}\mp@subsup{\pi}{k}{}
            by auto
        }
            ultimately have (Operator k (index ?ops op))+
        { {(Operator k (index ?ops op))+
            |op.op }\in\mathrm{ set ?ops }\wedgev\in\mathrm{ set (delete-effects-of op) }
        using v-is-delete-effect
        by blast
    then have (Operator k (index ?ops op))+}\in
        using C-is
        by auto
    moreover have \mathcal{A (Operator k (index ?ops op))}
        using assms(2) k-lt-length-\tau-minus-one op-in-\pi
        by blast
    ultimately show ?thesis
        unfolding clause-semantics-def
        by force
    next
    case B
    then obtain op
        where op-in-\pi}\mp@subsup{\pi}{k}{}:op\in\operatorname{set}(\pi!k
            and v-is-add-effect:v set (add-effects-of op)..
    then have }\neg(\existsop\in\operatorname{set}(\pi!k).v\in\operatorname{set}(\mathrm{ delete-effects-of op))
    using all-operator-effects-consistent are-all-operator-effects-consistent-set
        by fast
    then have execute-parallel-operator (?\tau !k)(\pi!k)v=Some True
```

```
using execute-parallel-operator-positive-effect-if[OF all-operators-applicable
                all-operator-effects-consistent op-in-\pi}\mp@subsup{\pi}{k}{}v\mathrm{ v-is-add-effect]
            by blast
            moreover have (?\tau !Suc k)v= execute-parallel-operator (?\tau!k) (\pi!k)
v
            using trace-parallel-plan-step-effect-is[OF k-lt-length-\tau-minus-one]
            by simp
    ultimately have \mathcal{A (State (Suc k) (index ?vs v))}
        using assms(3) k-lt-length-\tau-minus-one
        by simp
    thus ?thesis
        using C-is
        unfolding clause-semantics-def
        by simp
next
        case C
        show ?thesis
        - We split on cases for (?\tau !k) v=Some True here to avoid having to
proof (?\tau ! k) v\not= None.
    proof (cases (?\tau!k)v=Some True)
        case True
        {
            have (?\tau !Suc k) = execute-parallel-operator (?\tau ! k) ( }\pi!k
                    using trace-parallel-plan-step-effect-is[OF k-lt-length-\tau-minus-one].
            then have (?\tau!Suc k)v=(?\tau!k)v
                using execute-parallel-operator-no-effect-if C
                by fastforce
            then have (?\tau!Suc k)v=Some True
                using True
                by argo
            hence \mathcal{A (State (Suc k) (index ?vs v))}
                using assms(3) k-lt-length-\tau-minus-one
                by fastforce
        }
        thus ?thesis
            using C-is
                unfolding clause-semantics-def
                by fastforce
    next
        case False
        then have }\neg\mathcal{A}(\mathrm{ State }k(\mathrm{ index ?vs v))
            using assms(3) k-lt-length-\tau-minus-one
            by simp
        thus ?thesis
            using C-is
            unfolding clause-semantics-def
            by fastforce
    qed
qed
```

```
    next
        case k-gte-length-\tau-minus-one: False
        show ?thesis
        proof (cases \mathcal{A (State (length ?\tau - 1) (index ?vs v)))}
            case True
        {
            have length ? }\tau\leq\mathrm{ Suc k and Suc k < ?t + 1
                    using k-gte-length-\tau-minus-one k-in
                    by fastforce+
                    then have \mathcal{A (State (Suc k) (index ?vs v)) =\mathcal{A (State (length ?\tau - 1)}}\mathbf{~}=\mp@code{l}
(index ?vs v))
                using assms(4)
                by blast
                    hence \mathcal{A (State (Suc k) (index ?vs v))}
                    using True
                by blast
            }
            thus ?thesis
                    using C-is
                    unfolding clause-semantics-def
                    by fastforce
        next
            case False
            {
            have }\mathcal{A}(\mathrm{ State k (index ?vs v))}=\mathcal{A}(\mathrm{ State (length ? }\tau-1)(\mathrm{ index ?vs v))
                    proof (cases k= length ?\tau - 1)
                        case False
                        then have length ? }\tau\leqk\mathrm{ and k<?t + 1
                    using k}k\mathrm{ -gte-length-}\tau\mathrm{ -minus-one k-in
                    by fastforce+
                    thus ?thesis
                    using assms(4)
                    by blast
                    qed blast
                hence }\neg\mathcal{A}(\mathrm{ State k (index ?vs v))
                    using False
                by blast
            }
            thus ?thesis
                using C-is
                unfolding clause-semantics-def
                by simp
            qed
        qed
qed
lemma encode-problem-parallel-complete-iv:
    fixes \Pi::'a strips-problem
```

```
assumes is-valid-problem-strips \Pi
    and is-parallel-solution-for-problem \Pi\pi
    and }\forallk\mathrm{ op. }k<\mathrm{ length (trace-parallel-plan-strips ((П) }\mp@subsup{)}{I}{\prime})\pi)-
        \longrightarrow \mathcal { A } ( \text { Operator k (index (strips-problem.operators-of П) op))} = ( o p \in ~ s e t
(\pi!k))
    and \forallvk.k< length (trace-parallel-plan-strips ((\Pi) I) \pi)
        \longrightarrow ( \mathcal { A } \text { (State k (index (strips-problem.variables-of П) v))}
        \longleftrightarrow (trace-parallel-plan-strips }((\Pi\mp@subsup{)}{I}{})\pi!k)v=Some True
    and \forallvl.l\geq length (trace-parallel-plan-strips ((\Pi\mp@subsup{)}{I}{})\pi)\wedgel<length \pi+1
        \longrightarrow \mathcal { A ~ ( S t a t e ~ l ~ ( i n d e x ~ ( s t r i p s - p r o b l e m . v a r i a b l e s - o f ~ \Pi ) ~ v ) ) }
        = \mathcal{A (State}
            (length (trace-parallel-plan-strips ((\Pi)
            (index (strips-problem.variables-of \Pi) v))
    shows }\mathcal{A}\models\mathrm{ encode-all-frame-axioms }\Pi\mathrm{ (length }\pi\mathrm{ )
proof -
    let ? }\mp@subsup{\Phi}{F}{}=\mathrm{ encode-all-frame-axioms П (length }\pi\mathrm{ )
    let ?vs = strips-problem.variables-of \Pi
        and ?ops = strips-problem.operators-of \Pi
        and ?t = length \pi
    let ?A = \bigcup (\bigcup (k,v) \in{0..<?t} \times set ((\Pi)\mathcal{V}).
        {{{(State k (index ?vs v))}\mp@subsup{)}{}{+},(\mathrm{ State (Suc k) (index ?vs v))}\mp@subsup{)}{}{-1}
            \cup \{ ( \text { Operator k (index ?ops op))+}
                |op.op set ((\Pi\mp@subsup{)}{\mathcal{O}}{})\wedgev\in\operatorname{set}(add-effects-of op) }}})
    and ?B=\bigcup(\bigcup(k,v)\in{0..<?t} }\times\operatorname{set}((\Pi)\mathcal{V})
            {{{(State k (index ?vs v))}\mp@subsup{)}{}{-1},(\mathrm{ State (Suc k) (index ?vs v))+ }
                { (Operator k (index ?ops op))+
                    |op.op }\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{)})\wedgev\in\operatorname{set}(\mathrm{ delete-effects-of op) }}})
    have cnf-\Phi
        using cnf-of-encode-all-frame-axioms-structure
        by (simp add: cnf-of-encode-all-frame-axioms-structure)
    {
    fix C
    assume C\incnf ? }\mp@subsup{\Phi}{F}{
    then consider (C-in-A) C\in?A
        | (C-in-B) C ? ?B
        using Un-iff[of C ?A ?B] cnf-\Phi क्
        by argo
    hence clause-semantics \mathcal{A C}
        proof (cases)
            case C-in-A
            then show ?thesis
                                    using encode-problem-parallel-complete-iv-a[OF assms(2, 3, 4, 5) C-in-A]
                    by blast
        next
            case C-in-B
            then show ?thesis
                using encode-problem-parallel-complete-iv-b[OF assms(2, 3, 4, 5) C-in-B]
                    by blast
```

qed
\}
thus ?thesis
using encode-frame-axioms-is-cnf is-nnf-cnf cnf-semantics
unfolding cnf-semantics-def
by blast
qed
lemma valuation-for-operator-variables-is:
fixes $\Pi$ :: 'a strips-problem
assumes is-parallel-solution-for-problem $\Pi \pi$
and $k<$ length (trace-parallel-plan-strips $\left.\left((\Pi)_{I}\right) \pi\right)-1$
and $o p \in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)$
shows valuation-for-operator-variables $\Pi \pi$ (trace-parallel-plan-strips $\left.\left((\Pi)_{I}\right) \pi\right)$
(Operator $k$ (index (strips-problem.operators-of $\Pi$ ) op))
$=(o p \in \operatorname{set}(\pi!k))$
proof -
let ?ops $=$ strips-problem.operators-of $\Pi$
and $? \tau=$ trace-parallel-plan-strips $\left((\Pi)_{I}\right) \pi$
let $? v=$ Operator $k$ (index ?ops op)
and ?Op $=\{$ Operator $k$ (index ?ops op)
$\mid k$ op. $k \in\{0 . .<$ length ? $\left.\tau-1\} \wedge o p \in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)\right\}$
let ?l $=\operatorname{concat}(\operatorname{map}(\lambda k . \operatorname{map}($ Pair $k)(\pi!k))[0 . .<$ length ? $\tau-1])$
and ? $f=\lambda x$. Operator $(f s t x)($ index ?ops $($ snd $x))$

- show that our operator construction function is injective on set (concat (map
( $\lambda k$. map (Pair $k)(\pi!k))[0 . .<$ length ? $\tau-1]))$.
have $k$-in: $k \in\{0 . .<$ length $? \tau-1\}$
using assms(2)
by fastforce
\{
\{
fix $k k^{\prime} o p o p^{\prime}$
assume $k$-op-in: $(k, o p) \in$ set ?l and $k^{\prime}$-op'-in: $\left(k^{\prime}, o p^{\prime}\right) \in$ set ?l
have Operator $k$ (index ?ops op) $=$ Operator $k^{\prime}\left(\right.$ index ?ops op $\left.{ }^{\prime}\right) \longleftrightarrow(k, o p)$ $=\left(k^{\prime}, o p^{\prime}\right)$
proof (rule iffI)
assume index-op-is-index-op': Operator $k$ (index ?ops op) $=$ Operator $k^{\prime}$
(index ?ops op ${ }^{\prime}$ )
then have $k$-is- $k^{\prime}: k=k^{\prime}$
by fast
moreover \{
have $k^{\prime}-l t: k^{\prime}<$ length ? $\tau-1$
using $k^{\prime}$-op ${ }^{\prime}$-in
by fastforce
have op-in: op $\in \operatorname{set}(\pi!k)$
using $k$-op-in

```
            by force
            then have oo'-in:op' }\in\mathrm{ set ( }\pi!k
            using }\mp@subsup{k}{}{\prime}\mathrm{ -op'-in k-is-k'
            by auto
            {
            have length-\tau-gt-1: length ?\tau > 1
                using assms(2)
                by linarith
            have length ?\tau - Suc 0 \leq length \pi + 1 - Suc 0
                using length-trace-parallel-plan-strips-lte-length-plan-plus-one
                using diff-le-mono
                by blast
            then have length ? \tau - 1\leq length \pi
                by fastforce
            then have }\mp@subsup{k}{}{\prime}<l=length 
                using length-\tau-gt-1 k'-lt
                by linarith
            hence }\pi!\mp@subsup{k}{}{\prime}\in\mathrm{ set }
                by simp
            }
            moreover have op \in set ?ops and op' \in set ?ops
                    using is-parallel-solution-for-problem-operator-set[OF assms(1)] op-in
op'-in k-is-k'
                    calculation
                    by auto
            ultimately have op=op'
                    using index-op-is-index-op'
                    by force
            }
            ultimately show (k,op)=( k',o\mp@subsup{p}{}{\prime})
                by blast
            qed fast
    }
    hence inj-on ?f (set ?l)
        unfolding inj-on-def fst-def snd-def
        by fast
    } note inj-on-f-set-l = this
    {
    have set?l = \bigcup (set'set (map ( \lambdak. map (Pair k) (\pi!k)) [0..<length ?\tau -
1]))
            using set-concat
            by metis
            also have ... = \bigcup(set'(\lambdak. map (Pair k) (\pi!k))'{0..<length ?\tau - 1})
            by force
    also have ... = \bigcup ((\lambdak. (Pair k)' set ( }\pi!k))'{0..<length ?\tau - 1}
    by force
```

```
    also have ... = \bigcup((\lambdak.{(k,op)|op.op \in set (\pi!k) })'{0..<length ?\tau -
1})
            by blast
    also have ... = \bigcup({{(k,op)}|kop. k\in{0..<length?\tau - 1} ^op \in set
( }\pi!k)}
        by blast
    finally have set ?l = \bigcup ((\lambda(k,op).{(k,op) })
        '{(k,op).k\in{0..<length ?\tau - 1} ^op 就 ( }\pi!k)}
        using setcompr-eq-image[of \lambda(k,op). {(k,op)} -]
        by auto
} note set-l-is = this
{
    have Operator k (index ?ops op) \in?Op
        using assms(3) k-in
        by blast
    hence valuation-for-operator-variables \Pi \pi ?\tau ?v
        = foldr ( }\lambda(k,\mathrm{ op ) A. }\mathcal{A}(\mathrm{ Operator }k\mathrm{ (index ?ops op) := True)) ?l }\mp@subsup{\mathcal{A}}{0}{0}\mathrm{ ?v
        unfolding valuation-for-operator-variables-def override-on-def Let-def
        by auto
} note nb=this
show ?thesis
    proof (cases op \in set ( }\pi!k)
    case True
    moreover have k-op-in: (k,op)\in set ?l
        using set-l-is k-in calculation
        by blast
    - There is some problem with the pattern match in the lambda in fact, sow
we have to do some extra work to convince Isabelle of the truth of the statement.
    moreover {
        let ?g = \lambda-. True
        thm foldr-fun-upd[OF inj-on-f-set-l k-op-in]
        have ?v = Operator (fst (k,op)) (index ?ops (snd (k,op)))
            by simp
        moreover have ( }\lambda(k,op)\mathcal{A}.\mathcal{A}(\mathrm{ Operator }k\mathrm{ (index ?ops op) := True))
                        =(\lambdax\mathcal{A}.\mathcal{A}(Operator (fst x) (index ?ops (snd x)):= True))
                by fastforce
        moreover have foldr ( }\lambdax\mathcal{A}.\mathcal{A}(\mathrm{ Operator (fst x) (index ?ops (snd x)) :=
?g (x))
            ?l }\mp@subsup{\mathcal{A}}{0}{}(\mathrm{ Operator (fst (k,op)) (index ?ops (snd (k,op)))) = True
            unfolding foldr-fun-upd[OF inj-on-f-set-l k-op-in]..
        ultimately have valuation-for-operator-variables \Pi \pi ?\tau ?v = True
            using nb
            by argo
}
thus ?thesis
            using True
            by blast
```

```
    next
    case False
    {
        have (k,op) & set ?l
        using False set-l-is
        by fast
    moreover {
            fix }\mp@subsup{k}{}{\prime
            assume ( }\mp@subsup{k}{}{\prime},op')\in set ?l 
                and ?f ( }\mp@subsup{k}{}{\prime},o\mp@subsup{p}{}{\prime})=\mathrm{ ?f ( }k,op
            hence ( }\mp@subsup{k}{}{\prime},o\mp@subsup{p}{}{\prime})=(k,op
            using inj-on-f-set-l assms(3)
            by simp
    }
    ultimately have Operator k (index ?ops op) & ?f 'set ?l
                using image-iff
        by force
    } note operator-not-in-f-image-set-l = this
    {
        have }\mp@subsup{\mathcal{A}}{0}{}(\mathrm{ Operator k(index ?ops op))=False
        by simp
        moreover have (\lambda(k,op)\mathcal{A}.\mathcal{A}(\mathrm{ Operator k (index ?ops op) := True))}
            = (\lambdax\mathcal{A}.\mathcal{A}(Operator (fst x)(index ?ops (snd x)):= True))
        by fastforce
    ultimately have foldr ( }\lambda(k,op)\mathcal{A}.\mathcal{A}(Operator k (index ?ops op) := True)
?l }\mp@subsup{\mathcal{A}}{0}{}\mathrm{ ?v = False
                using foldr-fun-no-upd[OF inj-on-f-set-l operator-not-in-f-image-set-l, of
\lambda-. True }\mp@subsup{\mathcal{A}}{0}{}
                by presburger
    }
    thus ?thesis
    using nb False
    by blast
    qed
qed
lemma encode-problem-parallel-complete-vi-a:
    fixes \Pi :: 'a strips-problem
    assumes is-parallel-solution-for-problem \Pi \pi
        and k< length (trace-parallel-plan-strips ((П) I) \pi) - 1
    shows valuation-for-plan \Pi\pi (Operator k (index (strips-problem.operators-of \Pi)
op))
        =(op \inset (\pi!k))
proof -
    let ?vs = strips-problem.variables-of \Pi
        and ?ops = strips-problem.operators-of \Pi
```

and $? t=$ length $\pi$
and $?^{\tau}=$ trace-parallel-plan-strips $\left((\Pi)_{I}\right) \pi$
let ? $\mathcal{A}_{\pi}=$ valuation-for-plan $\Pi \pi$
and ${ }^{\prime} \mathcal{A}_{O}=$ valuation-for-operator-variables $\Pi \pi$ ? $\tau$
and ? Op $=\{$ Operator $k$ (index ?ops op) $\mid k$ op. $k \in\{0 . .<? t\} \wedge$ op $\in$ set ?ops
\}
and $? V=\{$ State $k($ index ?vs $v) \mid k v . k \in\{0 . .<? t+1\} \wedge v \in$ set ?vs $\}$
and $? v=$ Operator $k$ (index ?ops op)
\{
have length $? \tau \leq$ length $\pi+1$
using length-trace-parallel-plan-strips-lte-length-plan-plus-one.
then have length? $\tau-1 \leq$ length $\pi$
by simp
then have $k<$ ? $t$
using assms
by fastforce
\} note $k$-lt-length- $\pi=$ this
show ?thesis
proof $\left(\right.$ cases op $\left.\in \operatorname{set}\left((\Pi)_{\mathcal{O}}\right)\right)$
case True
\{
have ? $v \in ? O p$
using $k$-lt-length- $\pi$ True
by auto
hence ${ }^{?} \mathcal{A}_{\pi}{ }^{?} v=?{ }^{?} \mathcal{A}_{O}$ ? $v$
unfolding valuation-for-plan-def override-on-def Let-def
by force
\}
then show ?thesis
using valuation-for-operator-variables-is[OF $\operatorname{assms}(1,2) \operatorname{True}]$
by blast
next
case False
\{
\{

- We have $\neg$ index ? ops op < length ?ops due to the assumption that $\neg o p$
$\in$ set ?ops. Hence $\neg k \in\{0 . .<? t$ and therefore ?v $\notin ? O p$.
have ? $O p=(\lambda(k, o p)$. Operator $k($ index ?ops op $))$ ' $(\{0 . .<$ ? $t\} \times$ set ?ops $)$ by fast
moreover have $\neg$ index ?ops op $<$ length ?ops
using False
by $\operatorname{simp}$
ultimately have ?v $\notin ? O p$
by fastforce
\}
moreover have ?v $\notin ? V$
by force

```
        ultimately have ? }\mp@subsup{\mathcal{A}}{\pi}{}\mathrm{ ? v = 吿 ?v
            unfolding valuation-for-plan-def override-on-def
            by metis
            hence }\neg\mathrm{ ? }\mp@subsup{\mathcal{A}}{\pi}{}\mathrm{ ?v
            unfolding empty-valuation-def
            by blast
}
    moreover have ( }\pi!k)\in\mathrm{ set }
        using k-lt-length-\pi
        by simp
    moreover have op & set ( }\pi!k
        using is-parallel-solution-for-problem-operator-set[OF assms(1) calcula-
tion(2)] False
        by blast
        ultimately show ?thesis
        by blast
    qed
qed
lemma encode-problem-parallel-complete-vi-b:
    fixes \Pi :: 'a strips-problem
    assumes is-parallel-solution-for-problem \Pi \pi
        and l\geq length (trace-parallel-plan-strips ((\Pi) I) \pi) - 1
        and l< length }
    shows \negvaluation-for-plan \Pi \pi (Operator l (index (strips-problem.operators-of
П) op))
proof -
    let ?vs = strips-problem.variables-of \Pi
    and ?ops = strips-problem.operators-of \Pi
    and ?t = length \pi
    and ?\tau = trace-parallel-plan-strips ((\Pi) I) \pi
    let ? }\mp@subsup{\mathcal{A}}{\pi}{}=\mathrm{ valuation-for-plan }\Pi
    and ?}\mp@subsup{\mathcal{A}}{O}{}=\mathrm{ valuation-for-operator-variables }\Pi\pi\mathrm{ ? }
    and ?Op = { Operator k (index ?ops op) |k op.k\in{0..<?t}\wedgeop \in set?ops
}
    and ?Op' = { Operator k (index ?ops op) |kop. k\in{0..<length?\tau - 1} ^
op\in set ?ops }
    and ?V }={\mathrm{ State k (index ?vs v) |kv.k { {0..<?t + 1} ^v set?vs }
    and ?v = Operator l (index ?ops op)
show ?thesis
    proof (cases op }\in\operatorname{set}((\Pi\mp@subsup{)}{\mathcal{O}}{})
        case True
        {
            have ?v }\in\mathrm{ ?Op
                using assms(3) True
```

```
                by auto
```



```
                    unfolding valuation-for-plan-def override-on-def Let-def
                    by simp
        }
        moreover {
            have l}\not\in{0..<length? T - 1
            using assms(2)
            by simp
            then have ?v & ?Op'
            by blast
            hence ? }\mp@subsup{\mathcal{A}}{O}{}?v=\mp@subsup{\mathcal{A}}{0}{}\mathrm{ ? v
                    unfolding valuation-for-operator-variables-def override-on-def
            by meson
        }
        ultimately have }\neg\mathrm{ ? }\mp@subsup{\mathcal{A}}{\pi}{}\mathrm{ ?v
            unfolding empty-valuation-def
            by blast
        }
        then show ?thesis
            by blast
        next
    case False
    {
        {
            - We have \negindex ?ops op < length ?ops due to the assumption that \negop
set ?ops. Hence }\negk\in{0..<?t and therefore ?v \not\in?Op
    have ?Op = (\lambda(k,op). Operator k(index ?ops op))'({0..<?t} }\times\mathrm{ set ?ops )
            by fast
            moreover have }\neg\mathrm{ index ?ops op < length ?ops
            using False
            by simp
        ultimately have ?v & ?Op
            by fastforce
    }
    moreover have ?v & ?V
            by force
    ultimately have ? }\mp@subsup{\mathcal{A}}{\pi}{}\mathrm{ ? v = 吿 ?v
            unfolding valuation-for-plan-def override-on-def
            by metis
    hence }\neg\mathrm{ ? }\mp@subsup{\mathcal{A}}{\pi}{}\mathrm{ ?v
    unfolding empty-valuation-def
    by blast
}
thus ?thesis
    by blast
```


## qed

qed

- As a corollary from lemmas and we obtain the result that the constructed valuation $\mathcal{A} \equiv$ valuation-for-plan $\Pi \pi$ valuates SATPlan operator variables as false if they are not contained in any operator set $\pi!k$ for any time point $k<$ length $\pi$. corollary encode-problem-parallel-complete-vi-d:

```
    fixes \Pi :: 'variable strips-problem
    assumes is-parallel-solution-for-problem \Pi \pi
    and }k<\mathrm{ length }
    and op & set ( }\pi!k
    shows \negvaluation-for-plan \Pi\pi (Operator k (index (strips-problem.operators-of
```

П) $o p$ ))
using encode-problem-parallel-complete-vi-a[OF assms(1)] assms(3)
encode-problem-parallel-complete-vi-b[OF assms(1) - assms(2)] assms(3)
by (cases $k<$ length (trace-parallel-plan-strips $\left.\left((\Pi)_{I}\right) \pi\right)-1$; fastforce)
lemma list-product-is-nil-iff: List.product xs ys $=[] \longleftrightarrow x s=[] \vee y s=[]$
proof (rule iffI)
assume product-xs-ys-is-Nil: List.product xs ys $=[]$
show $x s=[] \vee y s=[]$
proof (rule ccontr)
assume $\neg(x s=[] \vee y s=[])$
then have $x s \neq[]$ and $y s \neq[$
by $\operatorname{simp}+$
then obtain $x x s^{\prime} y y s^{\prime}$ where $x s=x \# x s^{\prime}$ and $y s=y \# y s^{\prime}$
using list.exhaust
by metis
then have List.product xs ys $=(x, y) \#$ map (Pair $x) y s^{\prime} @$ List.product xs ${ }^{\prime}$
( $y \# y s^{\prime}$ )
by $\operatorname{simp}$
thus False
using product-xs-ys-is-Nil
by $\operatorname{simp}$
qed
next
assume $x s=[] \vee y s=[]$
thus List.product xs ys $=[]$
- First cases in the next two proof blocks follow from definition of List.product.
proof (rule disjE)
assume ys-is-Nil: ys $=[]$
show List.product xs ys = []
proof (induction xs)
case (Cons $x$ xs)
have List.product $(x \#$ xs $)$ ys $=$ map (Pair $x)$ ys @ List.product xs ys
by $\operatorname{simp}$
also have ... = [] @ List.product xs ys

```
            using Nil-is-map-conv ys-is-Nil
            by blast
            finally show ?case
                using Cons.IH
                by force
            qed auto
    qed simp
qed
- We keep the state abstract by requiring a function \(s\) which takes the index \(k\) and returns state. This makes the lemma cover both cases, i.e. dynamic (e.g. the \(k\)-th trace state) as well as static state (e.g. final trace state).
lemma valuation-for-state-variables-is:
assumes \(k \in\) set \(k s\)
and \(v \in\) set \(v s\)
shows foldr \((\lambda(k, v) \mathcal{A}\). valuation-for-state vs \((s k) k v \mathcal{A})\) (List.product ks vs) \(\mathcal{A}_{0}\)
(State \(k\) (index vs \(v\) ))
\(\longleftrightarrow(s k) v=\) Some True
proof -
let \(? v=\) State \(k\) (index vs \(v\) )
and ? ps \(=\) List.product ks vs
let ? \(\mathcal{A}=\) foldr \((\lambda(k, v) \mathcal{A}\). valuation-for-state vs \((s k) k v \mathcal{A})\) ?ps \(\mathcal{A}_{0}\)
and ?f \(=\lambda x\). State \((\) fst \(x)(\) index vs \((\) snd \(x))\)
and \(? g=\lambda x .(s(\) fst \(x))(\) snd \(x)=\) Some True
have \(n b_{1}:(k, v) \in\) set ?ps
using \(\operatorname{assms}(1,2)\) set-product
by \(\operatorname{simp}\)
```


## moreover \{

```
\{
fix \(x y\)
assume \(x\)-in-ps: \(x \in\) set \(? p s\) and \(y\)-in-ps: \(y \in\) set ?ps
and \(\neg\) (?f \(x=\) ? \(y \longrightarrow x=y\) )
then have \(f\) - \(x\)-is-f-y: ?f \(x=\) ?f \(y\) and \(x\)-is-not- \(y: x \neq y\)
by blast+
then obtain \(k^{\prime} k^{\prime \prime} v^{\prime} v^{\prime \prime}\)
where \(x\) - \(i s\) : \(x=\left(k^{\prime}, v^{\prime}\right)\)
and \(y\)-is: \(y=\left(k^{\prime \prime}, v^{\prime \prime}\right)\)
by fastforce
then consider \((A) k^{\prime} \neq k^{\prime \prime}\)
\(\mid(B) v^{\prime} \neq v^{\prime \prime}\)
using \(x\)-is-not-y
by blast
hence False
proof (cases)
case \(A\)
then have ?f \(x \neq\) ?f \(y\)
using \(x\)-is \(y\)-is
```

```
                by simp
            thus ?thesis
            using f-x-is-f-y
                by argo
            next
            case B
            have }\mp@subsup{v}{}{\prime}\in\mathrm{ set vs and }\mp@subsup{v}{}{\prime\prime}\in\mathrm{ set vs
            using x-in-ps x-is y-in-ps y-is set-product
            by blast+
            then have index vs v'}=\mathrm{ index vs v"
                using B
                by force
            then have ?f }x\not=\mathrm{ ?f }
                using x-is y-is
                by simp
            thus False
                using f-x-is-f-y
                by blast
            qed
    }
    hence inj-on ?f (set ?ps)
        using inj-on-def
        by blast
    } note n\mp@subsup{b}{2}{}= this
    {
        have foldr ( }\lambdax.valuation-for-state vs (s (fst x)) (fst x) (snd x)) 
        (List.product ks vs) \mathcal{A}
    (s (fst (k,v)) (snd (k,v)) = Some True)
        using foldr-fun-upd[OF n\mp@subsup{b}{2}{}}n\mp@subsup{b}{1}{}\mathrm{ , of ?g }\mp@subsup{\mathcal{A}}{0}{}
        by blast
    moreover have ( }\lambdax\mathrm{ . valuation-for-state vs (s (fst x)) (fst x) (snd x))
        =(\lambda(k,v).valuation-for-state vs (sk)kv)
        by fastforce
    ultimately have ?. A (?f (k,v))=?g(k,v)
    by simp
}
thus ?thesis
    by simp
qed
lemma encode-problem-parallel-complete-vi-c:
    fixes \Pi :: 'a strips-problem
    assumes is-valid-problem-strips \Pi
    and is-parallel-solution-for-problem \Pi\pi
    and k< length (trace-parallel-plan-strips ((\Pi\mp@subsup{)}{I}{})\pi)
    shows valuation-for-plan \Pi\pi (State k (index (strips-problem.variables-of \Pi) v))
    \longleftrightarrow ( \text { trace-parallel-plan-strips ((П) I) } \pi ! k ) v = \text { Some True}
proof -
```

```
    let ?vs = strips-problem.variables-of \Pi
    and ?ops = strips-problem.operators-of \Pi
    and ?\tau = trace-parallel-plan-strips ((\Pi) I) \pi
    let ?t = length }
    and ?t' = length ?\tau
    let ? }\mp@subsup{\mathcal{A}}{\pi}{}=\mathrm{ valuation-for-plan }\Pi
    and ?}\mp@subsup{\mathcal{A}}{V}{}=\mathrm{ valuation-for-state-variables }\Pi\pi\mathrm{ ? }
    and ?}\mp@subsup{\mathcal{A}}{O}{}=\mathrm{ valuation-for-state-variables }\Pi\pi\mathrm{ ? }
    and ?}\mp@subsup{\mathcal{A}}{1}{}=\mathrm{ foldr
        (\lambda(k,v)\mathcal{A}.valuation-for-state ?vs (?\tau ! k)kv\mathcal{A})
        (List.product [0..<?t'] ?vs) }\mp@subsup{\mathcal{A}}{0}{
    and ?Op = { Operator k (index ?ops op) |kop.k\in{0..<?t} ^op \in set ((\Pi) ( )
}
    and ?Op' = { Operator k (index ?ops op) | k op. k\in{0..<?t' - 1} ^op fet
((\Pi)
    and ?V = { State k (index ?vs v)|kv. k\in{0..<?t + 1} ^v\in set ((\Pi)\mathcal{V})}
    and ? V V = {State k (index ?vs v) |kv.k\in{0..<?t'} ^v\in\operatorname{set ((\Pi)v)}}
    and ? V 
}
    and ?v = State k (index ?vs v)
have v-notin-Op: ?v & ?Op
    by blast
have k-lte-length-\pi-plus-one: }k<\mathrm{ length }\pi+
    using less-le-trans length-trace-parallel-plan-strips-lte-length-plan-plus-one assms(3)
    by blast
show ?thesis
    proof (cases v\in set ((\Pi)\mathcal{V}))
        case True
        {
            {
            have ?v\in?V ?v & ?Op
                    using k-lte-length-\pi-plus-one True
                            by force+
                            hence ?. }\mp@subsup{\mathcal{A}}{\pi}{}\mathrm{ ?v = ? . }\mp@subsup{\mathcal{A}}{V}{}\mathrm{ ?v
                            unfolding valuation-for-plan-def override-on-def Let-def
                            by simp
            }
            moreover {
                    have ?v }\in
                            using assms(3) True
                            by fastforce+
                            hence ?. }\mp@subsup{\mathcal{A}}{V}{}\mathrm{ ?v = ? }\mp@subsup{\mathcal{A}}{1}{}\mathrm{ ?v
                            unfolding valuation-for-state-variables-def override-on-def Let-def
                    by force
            }
            ultimately have ? }\mp@subsup{\mathcal{A}}{\pi}{}\mathrm{ ? v = ? . }\mp@subsup{\mathcal{A}}{1}{}\mathrm{ ?v
                        by blast
```

```
        }
        moreover have k}\in\mathrm{ set [0..<?t']
            using assms(3)
            by simp
    moreover have v\in set (strips-problem.variables-of \Pi)
            using True
            by simp
            ultimately show ?thesis
            using valuation-for-state-variables-is[of k [0..<?t \]]
            by fastforce
        next
            case False
            {
            {
            have }\neg\mathrm{ index ?vs v< length ?vs
                using False index-less-size-conv
                by simp
            hence ?v & ?V
                by fastforce
            }
            then have }\neg\mathrm{ ? . }\mp@subsup{\mathcal{T}}{\pi}{}\mathrm{ ?v
            using v-notin-Op
        unfolding valuation-for-plan-def override-on-def empty-valuation-def Let-def
            variables-of-def operators-of-def
            by presburger
        }
        moreover have }\neg(?\tau!k)v=Some Tru
            using trace-parallel-plan-strips-none-if[of \Pi\pikv] assms(1, 2, 3) False
            unfolding initial-of-def
            by force
            ultimately show ?thesis
            by blast
        qed
qed
lemma encode-problem-parallel-complete-vi-f:
    fixes \Pi :: 'a strips-problem
    assumes is-valid-problem-strips \Pi
        and is-parallel-solution-for-problem \Pi\pi
        and l\geq length (trace-parallel-plan-strips ((\Pi) () ) \pi)
        and}l<length \pi+
    shows valuation-for-plan \Pi\pi (State l (index (strips-problem.variables-of \Pi) v))
    = valuation-for-plan \Pi \pi
            (State (length (trace-parallel-plan-strips ((\Pi) I) \pi) - 1)
            (index (strips-problem.variables-of \Pi) v))
proof -
```

```
    let ?vs = strips-problem.variables-of \Pi
    and ?ops = strips-problem.operators-of \Pi
    and ?}\tau=\mathrm{ trace-parallel-plan-strips ((П) I) }
    let ?t = length }
    and ?t' = length ?\tau
    let ?}\mp@subsup{\tau}{\Omega}{}=?\tau\tau!(?\mp@subsup{t}{}{\prime}-1
    and ?}\mp@subsup{\mathcal{A}}{\pi}{}=\mathrm{ valuation-for-plan }\Pi
    and ?}\mp@subsup{\mathcal{A}}{V}{}=\mathrm{ valuation-for-state-variables }\Pi\pi\mathrm{ ? }
    and ?}\mp@subsup{\mathcal{A}}{O}{}=\mathrm{ valuation-for-state-variables }\Pi\pi\mathrm{ ? }
    let ?. }\mp@subsup{\mathcal{A}}{2}{}=\mathrm{ foldr
    (\lambda(k,v)\mathcal{A.valuation-for-state (strips-problem.variables-of \Pi) ?}\mp@subsup{\tau}{\Omega}{}kv\mathcal{A})
    (List.product [?t'..<length \pi + 2] ?vs)
    \mathcal{A}
    and ?Op ={ Operator k(index ?ops op) | kop.k\in{0..<?t} ^op f set ((\Pi) ( ) 
}
    and ?Op' = { Operator k (index ?ops op) |k op.k\in{0..<?t' - 1} ^op fet
((\Pi)
```



```
    and ?V}\mp@subsup{V}{1}{}={\mathrm{ State k (index ?vs v) |kv.k { {0..<?t'} ^v set ((П)v) }
```



```
}
    and ?v = State l (index ?vs v)
have v-notin-Op: ?v & ?Op
    by blast
show ?thesis
    proof (cases v\in set ((\Pi)\mathcal{V}))
        case True
        {
            {
            have ?v\in ?V ?v & ?Op
                using assms(4) True
                    by force+
            hence ?. }\mp@subsup{\mathcal{A}}{\pi}{}\mp@subsup{}{}{?}v=? ?.\mathcal{A}V ?
                unfolding valuation-for-plan-def override-on-def Let-def
                by simp
            }
            moreover {
                    have ?v }\not\in?\mp@subsup{V}{1}{}?v\in?,\mp@subsup{V}{2}{
                    using assms(3, 4) True
                    by force+
            hence ?. }\mp@subsup{\mathcal{A}}{V}{}?v=\mathrm{ ? }\mp@subsup{\mathcal{A}}{2}{}\mathrm{ ? v
                    unfolding valuation-for-state-variables-def override-on-def Let-def
                    by auto
            }
            ultimately have ? }\mp@subsup{\mathcal{A}}{\pi}{}\mathrm{ ? v = ? ( }\mp@subsup{\mathcal{A}}{2}{}\mathrm{ ?v
                by blast
```

```
} note nb=this
moreover
{
    have l\in set [?t'..<?t + 2]
        using assms(3, 4)
        by auto
    hence ?. . 
        using valuation-for-state-variables-is[of l [?t'..<?t + 2]] True nb
        by fastforce
}
ultimately have ? '\mathcal{A}
    by fast
moreover {
    have 0<? ?t'
        using trace-parallel-plan-strips-not-nil
        by blast
    then have ? }\mp@subsup{t}{}{\prime}-1<?!\mp@subsup{t}{}{\prime
        using diff-less
        by presburger
}
ultimately show ?thesis
    using encode-problem-parallel-complete-vi-c[of--?t' - 1,OF assms(1, 2)]
    by blast
next
    case False
{
    have \neg index ?vs v<length ?vs
        using False index-less-size-conv
        by auto
        hence ?v & ?V
            by fastforce
    }
    then have \neg? ( \mathcal{ }}\mathrm{ ?v
        using v-notin-Op
    unfolding valuation-for-plan-def override-on-def empty-valuation-def Let-def
        variables-of-def operators-of-def
        by presburger
}
moreover {
    have 0<? ?'
        using trace-parallel-plan-strips-not-nil
        by blast
    then have ? }\mp@subsup{t}{}{\prime}-1<??\mp@subsup{t}{}{\prime
        by simp
}
moreover have }\neg((?\tau!(?\mp@subsup{t}{}{\prime}-1))v=Some True
        using trace-parallel-plan-strips-none-if[of - - ?t' - 1 v,OF - assms(2)
```

```
calculation(2)]
            assms(1) False
        by \(\operatorname{simp}\)
        ultimately show ?thesis
            using encode-problem-parallel-complete-vi-c[of--?t' - 1, OF \(\operatorname{assms}(1,2)]\)
            by blast
    qed
qed
```

Let now $\tau \equiv$ trace-parallel-plan-strips $I \pi$ be the trace of the plan $\pi, t \equiv$ length $\pi$, and $t^{\prime} \equiv$ length $\tau$.
Any model of the SATPlan encoding $\mathcal{A}$ must satisfy the following properties: 11

1. for all $k$ and for all $o p$ with $k<t^{\prime}-\left(1::^{\prime} a\right)$
```
\mathcal { A ~ ( O p e r a t o r ~ k ~ ( i n d e x ~ ( o p e r a t o r s - o f ~ \Pi ) ~ o p ) ) ~ = o p ~ \in ~ s e t ~ ( ~ } \pi ! k )
```

2. for all $l$ and for all $o p$ with $t^{\prime}-\left(1::^{\prime} a\right) \leq l$ and $l<l e n g t h ~ \pi$ we require
```
A (Operator l (index (operators-of \Pi) op))
```

3. for all $v$ and for all $k$ with $k<t^{\prime}$ we require

$$
\mathcal{A}(\text { State } k(\text { index }(\text { variables-of } \Pi) v)) \longrightarrow((\tau!k) v=\text { Some True })
$$

4. and finally for all $v$ and for all $l$ with $t^{\prime} \leq l$ and $l<t+\left(1::^{\prime} a\right)$ we require
```
\mathcal{A (State l (index (variables-of \Pi) v))}
```



Condition "1." states that the model must reflect operator activation for all operators in the parallel operator lists $\pi!k$ of the plan $\pi$ for each time step $k<t^{\prime}-\left(1::^{\prime} a\right)$ s.t. there is a successor state in the trace. Moreover " 3 ." requires that the model is consistent with the states reached during plan execution (i.e. the elements $\tau!k$ for $k<t^{\prime}$ of the trace $\tau$ ). Meaning that $\mathcal{A}$ (State $k\left(\operatorname{index}\left(\Pi_{\mathcal{V}}\right) v\right)$ ) for the SAT plan variable of every state variable $v$ at time point $k$ if and only if $(\tau!k) v=$ Some True for the corresponding state $\tau!k$ at time $k$ (and $\neg \mathcal{A}\left(\right.$ State $\left.k\left(\operatorname{index}\left(\Pi_{\mathcal{V}}\right) v\right)\right)$ otherwise).
The second respectively fourth condition cover early plan termination by negating operator activation and propagating the last reached state. Note

[^9]that in the state propagation constraint, the index is incremented by one compared to the similar constraint for operators, since operator activations are always followed by at least one successor state. Hence the last state in the trace has index length (trace-parallel-plan-strips $\left.\left(\Pi_{I}\right) \pi\right)-1$ and the remaining states take up the indexes to length $\pi+1$.
value stop

- To show completeness - i.e. every valid parallel plan $\pi$ corresponds to a model for the SATPlan encoding $\Phi \Pi$ (length $\pi$ )—, we simply split the conjunction defined by the encoding into partial encodings and show that the model satisfies each of them.

```
theorem
    encode-problem-parallel-complete:
    assumes is-valid-problem-strips \(\Pi\)
    and is-parallel-solution-for-problem \(\Pi \pi\)
    shows valuation-for-plan \(\Pi \pi \models \Phi \Pi\) (length \(\pi\) )
proof -
    let \(? t=\) length \(\pi\)
        and \(? I=(\Pi)_{I}\)
        and ? \(G=(\Pi)_{G}\)
        and ? \(\mathcal{A}=\) valuation-for-plan \(\Pi \pi\)
    have \(n b:\) ? \(G \subseteq_{m}\) execute-parallel-plan ?I \(\pi\)
        using assms(2)
        unfolding is-parallel-solution-for-problem-def
        by force
    have ? \(\mathcal{A} \models \Phi_{I} \Pi\)
        using encode-problem-parallel-complete-i[OF assms(1) nb]
            encode-problem-parallel-complete-vi-c[OF \(\operatorname{assms}(1,2)]\)
    by presburger
moreover have ? \(\mathcal{A} \vDash\left(\Phi_{G} \Pi\right)\) ? \(t\)
    using encode-problem-parallel-complete-ii[OF assms(1) nb]
        encode-problem-parallel-complete-vi-c[OF \(\operatorname{assms}(1,2)]\)
        encode-problem-parallel-complete-vi-f[OF \(\operatorname{assms}(1,2)]\)
    by presburger
moreover have ? \(\mathcal{A}=\) encode-operators \(\Pi\) ?t
    using encode-problem-parallel-complete-iii[OF assms(1) nb]
            encode-problem-parallel-complete-vi-a[OF assms(2)]
            encode-problem-parallel-complete-vi-b[OF assms(2)]
            encode-problem-parallel-complete-vi-c[OF \(\operatorname{assms}(1,2)]\)
    by presburger
moreover have ? \(\mathcal{A} \models\) encode-all-frame-axioms \(\Pi\) ?t
    using encode-problem-parallel-complete-iv[OF \(\operatorname{assms}(1,2)]\)
            encode-problem-parallel-complete-vi-a[OF assms(2)]
            encode-problem-parallel-complete-vi-c[OF \(\operatorname{assms}(1,2)]\)
            encode-problem-parallel-complete-vi-f[OF \(\operatorname{assms}(1,2)]\)
    by presburger
ultimately show ?thesis
    unfolding encode-problem-def SAT-Plan-Base.encode-problem-def
```

```
        encode-initial-state-def encode-goal-state-def
    by auto
qed
end
```

theory SAT-Plan-Extensions
imports SAT-Plan-Base
begin

## 8 Serializable SATPlan Encodings

A SATPlan encoding with exclusion of operator interference (see definition ??) can be defined by extending the basic SATPlan encoding with clauses

```
\neg ( \text { Atom (Operator k (index ops op } 1 \text { ))}
\vee \neg ( \text { Atom (Operator k (index ops op 2))}
```

for all pairs of distinct interfering operators $o p_{1}$, $o p_{2}$ for all time points $k<$ $t$ for a given estimated plan length $t$. Definitions ?? and ?? implement the encoding for operator pairs resp. for all interfering operator pairs and all time points.

```
definition encode-interfering-operator-pair-exclusion
    :: 'variable strips-problem
    \(\Rightarrow\) nat
    \(\Rightarrow\) 'variable strips-operator
    \(\Rightarrow\) 'variable strips-operator
    \(\Rightarrow\) sat-plan-variable formula
    where encode-interfering-operator-pair-exclusion \(\Pi k o p_{1} o p_{2}\)
    \(\equiv\) let ops \(=\) operators-of \(\Pi\) in
    \(\neg\left(\right.\) Atom \(\left(\right.\) Operator \(k\left(\right.\) index ops op \(\left.\left.\left.p_{1}\right)\right)\right)\)
    \(\vee \neg(\) Atom \((\) Operator \(k(\) index ops op 2\()))\)
```

definition encode-interfering-operator-exclusion
:: 'variable strips-problem $\Rightarrow$ nat $\Rightarrow$ sat-plan-variable formula
where encode-interfering-operator-exclusion $\Pi t \equiv$ let
ops $=$ operators-of $\Pi$
; interfering $=$ filter $\left(\lambda\left(o p_{1}, o p_{2}\right)\right.$. index ops op $1 \neq$ index ops op ${ }_{2}$
$\wedge$ are-operators-interfering op $\boldsymbol{p}_{1}$ op ${ }_{2}$ ) (List.product ops ops)
in foldr $(\wedge)$ [encode-interfering-operator-pair-exclusion $\Pi k o p_{1} o p_{2}$.
$\left(o p_{1}, o p_{2}\right) \leftarrow$ interfering, $\left.k \leftarrow[0 . .<t]\right](\neg \perp)$

A SATPlan encoding with interfering operator pair exclusion can now be defined by simplying adding the conjunct encode-interfering-operator-exclusion $\Pi t$ to the basic SATPlan encoding.
definition encode-problem-with-operator-interference-exclusion

```
:: 'variable strips-problem }=>\mathrm{ nat }=>\mathrm{ sat-plan-variable formula
( }\mp@subsup{\Phi}{\forall}{}--52
where encode-problem-with-operator-interference-exclusion \Pit
     encode-initial-state \Pi
        ^(encode-operators \Pi t
    ^(encode-all-frame-axioms \Pit
    \wedge (encode-interfering-operator-exclusion \Pi t
    \wedge(encode-goal-state \Pit))))
```

- Immediately proof the sublocale proposition for strips in order to gain access to definitions and lemmas.
lemma cnf-of-encode-interfering-operator-pair-exclusion-is-i[simp]:
cnf (encode-interfering-operator-pair-exclusion $\left.\Pi k o p_{1} o p_{2}\right)=\{\{$
(Operator $k$ (index (strips-problem.operators-of $\Pi$ ) op $\left.\left.)_{1}\right)\right)^{-1}$
, (Operator $k$ (index (strips-problem.operators-of $\Pi$ ) $\left.\left.\left.\left.o p_{2}\right)\right)^{-1}\right\}\right\}$
proof -
let ?ops $=$ strips-problem.operators-of $\Pi$
have cnf (encode-interfering-operator-pair-exclusion $\left.\Pi k o p_{1} o p_{2}\right)$
$=\operatorname{cnf}\left(\neg\left(\right.\right.$ Atom $\left(\right.$ Operator $k\left(\right.$ index ?ops op $\left.\left.\left.p_{1}\right)\right)\right) \vee \neg$ (Atom (Operator $k$ (index ?ops op 2 ))))
unfolding encode-interfering-operator-pair-exclusion-def
by metis
also have $\ldots=\{C \cup D \mid C D$.
$C \in \operatorname{cnf}\left(\neg\left(\right.\right.$ Atom $\left(\right.$ Operator $k\left(\right.$ index ?ops op $\left.\left.\left.\left.{ }_{1}\right)\right)\right)\right)$
$\wedge D \in \operatorname{cnf}\left(\neg\left(\right.\right.$ Atom $\left(\right.$ Operator $k\left(\right.$ index ?ops op $\left.\left.\left.\left.\left.{ }_{2}\right)\right)\right)\right)\right\}$
by $\operatorname{simp}$
finally show ?thesis
by auto
qed
lemma cnf-of-encode-interfering-operator-exclusion-is-ii[simp]: set [encode-interfering-operator-pair-exclusion $\Pi k$ op $p_{1} o p_{2}$.
$\left(o p_{1}, o p_{2}\right) \leftarrow$ filter $\left(\lambda\left(o p_{1}, o p_{2}\right)\right.$.
index (strips-problem.operators-of $\Pi$ ) op $p_{1} \neq$ index (strips-problem.operators-of
П) $o p_{2}$
$\wedge$ are-operators-interfering op ${ }_{1} o p_{2}$ )
(List.product (strips-problem.operators-of $\Pi$ ) (strips-problem.operators-of
П))
$, k \leftarrow[0 . .<t]]$
$=\left(\bigcup\left(o p_{1}, o p_{2}\right)\right.$
$\in\left\{\left(o p_{1}, o p_{2}\right) \in \operatorname{set}(\right.$ operators-of $\Pi) \times$ set (operators-of $\Pi$ ).
index (strips-problem.operators-of $\Pi$ ) op $p_{1} \neq$ index (strips-problem.operators-of
П) $o p_{2}$
$\wedge$ are-operators-interfering $\left.o p_{1} o p_{2}\right\}$.
( $\lambda k$. encode-interfering-operator-pair-exclusion $\Pi k o p_{1} o p_{2}$ )' $\{0 . .<t\}$ )
proof -

```
let ?ops = strips-problem.operators-of \Pi
```

let ?interfering $=$ filter $\left(\lambda\left(o p_{1}, o p_{2}\right)\right.$. index ?ops op $p_{1} \neq$ index ?ops op $p_{2}$
$\wedge$ are-operators-interfering op op $_{2}$ ) (List.product ?ops ?ops)
let ?fs $=\left[\right.$ encode-interfering-operator-pair-exclusion $\Pi k o p_{1} o p_{2}$.
$\left(o p_{1}, o p_{2}\right) \leftarrow$ ? interfering, $\left.k \leftarrow[0 . .<t]\right]$
have set ? fs $=\bigcup$ (set
' $\left(\lambda\left(o p_{1}, o p_{2}\right)\right.$. map ( $\lambda k$. encode-interfering-operator-pair-exclusion $\left.\Pi k o p_{1} o p_{2}\right)$
$[0 . .<t])$
' (set (filter $\left(\lambda\left(o p_{1}, \text { op }\right)_{2}\right)$. index ?ops op $p_{1} \neq$ index ?ops op $p_{2} \wedge$ are-operators-interfering
$\left.o p_{1} o p_{2}\right)$
(List.product ?ops ?ops))))
unfolding set-concat set-map
by blast
- TODO slow.
also have $\ldots=\bigcup\left(\left(\lambda\left(o p_{1}, o p_{2}\right)\right.\right.$.
set (map ( $\lambda k$. encode-interfering-operator-pair-exclusion $\left.\left.\Pi k o p_{1} o p_{2}\right)[0 . .<t]\right)$ )
' (set (filter $\left(\lambda\left(o p_{1}, o p_{2}\right)\right.$. index ?ops op ${ }_{1} \neq$ index ?ops op ${ }_{2} \wedge$ are-operators-interfering
$\left.o p_{1} o p_{2}\right)$
(List.product ?ops ?ops))))
unfolding image-comp [of
set $\lambda\left(o p_{1}, o p_{2}\right) . \operatorname{map}\left(\lambda k\right.$. encode-interfering-operator-pair-exclusion $\Pi k o p_{1}$
$\left.\left.o p_{2}\right)[0 . .<t]\right]$
comp-apply
by fast
also have $\ldots=\bigcup\left(\left(\lambda\left(o p_{1}, o p_{2}\right)\right.\right.$.
( $\lambda k$. encode-interfering-operator-pair-exclusion $\Pi k o p_{1} o p_{2}$ )' $\left.\{0 . .<t\}\right)$
' (set (filter $\left(\lambda\left(o p_{1}, o p_{2}\right)\right.$. index ?ops op $p_{1} \neq$ index ?ops op ${ }_{2} \wedge$ are-operators-interfering
$\left.o p_{1} o p_{2}\right)$
(List.product ?ops ?ops))))
unfolding set-map $[$ of $-[0 . .<t]]$ atLeastLessThan-upt $[$ of $0 t]$
by blast
also have $\ldots=\bigcup\left(\left(\lambda\left(o p_{1}, o p_{2}\right)\right.\right.$.
( $\lambda k$. encode-interfering-operator-pair-exclusion $\Pi k o p_{1} o p_{2}$ )' $\{0 . .<t\}$ )
' (Set.filter $\left(\lambda\left(o p_{1}\right.\right.$, op 2$)$. index ?ops op $1 \neq$ index ?ops op ${ }_{2} \wedge$ are-operators-interfering
$\left.o p_{1} o p_{2}\right)$
(set (List.product ?ops ?ops))))
unfolding set-filter[of $\lambda\left(o p_{1}, o p_{2}\right)$. are-operators-interfering op ${ }_{1} o p_{2}$ List.product
?ops ?ops]
by force
- TODO slow.
finally show ?thesis
unfolding operators-of-def set-product[of ?ops ?ops]
by fastforce
qed
lemma cnf-of-encode-interfering-operator-exclusion-is-iii $[$ simp $]$ :
fixes $\Pi$ :: 'variable strips-problem
shows cnf'set [encode-interfering-operator-pair-exclusion $\Pi k$ op $p_{1} o p_{2}$.
$\left(o p_{1}, o p_{2}\right) \leftarrow$ filter $\left(\lambda\left(o p_{1}, o p_{2}\right)\right.$.
index (strips-problem.operators-of $\Pi$ ) op $p_{1} \neq$ index (strips-problem.operators-of
П) $o p_{2}$
$\wedge$ are-operators-interfering op ${ }_{1} o p_{2}$ )
(List.product (strips-problem.operators-of $\Pi$ ) (strips-problem.operators-of
П))

$$
\begin{aligned}
& , k \leftarrow[0 . .<t]] \\
= & \left(\bigcup\left(o p_{1}, o p_{2}\right)\right. \\
& \in\left\{\left(o p_{1}, o p_{2}\right) \in \text { set (strips-problem.operators-of } \Pi\right) \times \text { set }(\text { strips-problem.operators-of }
\end{aligned}
$$

П).
index (strips-problem.operators-of $\Pi$ ) op $p_{1} \neq$ index (strips-problem.operators-of
П) $o p_{2}$
$\wedge$ are-operators-interfering op $\left.{ }_{1} o p_{2}\right\}$.
$\left\{\left\{\left\{(\text { Operator } k \text { (index (strips-problem.operators-of } \Pi) o p_{1}\right)\right)^{-1}\right.$
, (Operator $k$ (index (strips-problem.operators-of $\Pi$ ) op $\left.\left.\left.)_{2}\right)^{-1}\right\}\right\} \mid k . k \in$
$\{0 . .<t\}\})$
proof -
let ?ops $=$ strips-problem.operators-of $\Pi$
let ?interfering $=$ filter $\left(\lambda\left(o p_{1}, o p_{2}\right)\right.$. index ?ops op ${ }_{1} \neq$ index ?ops op ${ }_{2}$
$\wedge$ are-operators-interfering op op $_{2}$ ) (List.product ?ops ?ops)
let ?fs $=\left[\right.$ encode-interfering-operator-pair-exclusion $\Pi k o p_{1} o p_{2}$.
$\left(o p_{1}, o p_{2}\right) \leftarrow$ ? interfering, $\left.k \leftarrow[0 . .<t]\right]$
have cnf'set ? $f_{s}=c n f$ ' $\left(\bigcup\left(o p_{1}, o p_{2}\right) \in\left\{\left(o p_{1}, o p_{2}\right)\right.\right.$.
$\left(o p_{1}, o p_{2}\right) \in \operatorname{set}($ operators-of $\Pi) \times$ set (operators-of $\left.\Pi\right) \wedge$ index ?ops op $p_{1} \neq$ index ?ops op ${ }_{2}$
$\wedge$ are-operators-interfering op $\left.{ }_{1} o p_{2}\right\}$.
( $\lambda k$. encode-interfering-operator-pair-exclusion $\Pi k o p_{1} o p_{2}$ )' $\{0 . .<t\}$ )
unfolding cnf-of-encode-interfering-operator-exclusion-is-ii
by blast
also have $\ldots=\left(\bigcup\left(o p_{1}, o p_{2}\right) \in\left\{\left(o p_{1}, o p_{2}\right)\right.\right.$.
$\left(o p_{1}, o p_{2}\right) \in \operatorname{set}($ operators-of $\Pi) \times$ set (operators-of $\left.\Pi\right) \wedge$ index ?ops op $p_{1} \neq$ index ?ops op ${ }_{2}$
$\wedge$ are-operators-interfering op $\left.o p_{2}\right\}$.
$\left(\lambda k . c n f\left(\right.\right.$ encode-interfering-operator-pair-exclusion $\left.\left.\Pi k o p_{1} o p_{2}\right)\right)$ ' $\left.\{0 . .<t\}\right)$
unfolding image-Un image-comp comp-apply
by blast
also have $\ldots=\left(\bigcup\left(o p_{1}, o p_{2}\right) \in\left\{\left(o p_{1}, o p_{2}\right)\right.\right.$.
$\left(o p_{1}, o p_{2}\right) \in \operatorname{set}($ operators-of $\Pi) \times$ set (operators-of $\left.\Pi\right) \wedge$ index ?ops op $p_{1} \neq$ index ?ops op 2
$\wedge$ are-operators-interfering op $\left.{ }_{1} o p_{2}\right\}$.
$\left(\lambda k\right.$. $\left.\left\{\left\{(\text { Operator } k(\text { index ?ops op } 1))^{-1},\left(\text { Operator } k\left(\text { index ?ops op }{ }_{2}\right)\right)^{-1}\right\}\right\}\right)$ ' $\{0 . .<t\})$
by $\operatorname{simp}$
also have $\ldots=\left(\bigcup\left(o p_{1}, o p_{2}\right) \in\left\{\left(o p_{1}, o p_{2}\right)\right.\right.$.
$\left(o p_{1}, o p_{2}\right) \in \operatorname{set}($ operators-of $\Pi) \times$ set (operators-of $\left.\Pi\right) \wedge$ index ?ops op $p_{1} \neq$ index ?ops op ${ }_{2}$
$\wedge$ are-operators-interfering op $\left.{ }_{1} o p_{2}\right\}$.
$\left(\lambda k\right.$. $\left.\left\{\left\{(\text { Operator } k(\text { index ? ops op } 1))^{-1},\left(\text { Operator } k\left(\text { index ?ops op }{ }_{2}\right)\right)^{-1}\right\}\right\}\right)$

$$
\{k \mid k . k \in\{0 . .<t\}\})
$$

by blast

- TODO slow.
finally show ?thesis
unfolding operators-of-def setcompr-eq-image $[o f-\lambda k . k \in\{0 . .<t\}]$
by force
qed
lemma cnf-of-encode-interfering-operator-exclusion-is:
cnf (encode-interfering-operator-exclusion $\Pi t)=\bigcup\left(\bigcup\left(o p_{1}, o p_{2}\right)\right.$
$\in\left\{\left(o p_{1}, o p_{2}\right) \in \operatorname{set}\right.$ (operators-of $\left.\Pi\right) \times$ set (operators-of $\Pi$ ).
index (strips-problem.operators-of $\Pi$ ) op $p_{1} \neq$ index (strips-problem.operators-of
П) $o p_{2}$
$\wedge$ are-operators-interfering $\left.o p_{1} o p_{2}\right\}$.
$\left\{\left\{\left\{\left(\text { Operator } k \text { (index (strips-problem.operators-of } \Pi \text { ) op } p_{1}\right)\right)^{-1}\right.\right.$
, (Operator $k$ (index (strips-problem.operators-of $\Pi$ ) op 2 ) $\left.\left.)^{-1}\right\}\right\} \mid k . k \in$
$\{0 . .<t\}\}$ )
proof -
let ?ops $=$ strips-problem.operators-of $\Pi$
let ?interfering $=$ filter $\left(\lambda\left(o p_{1}, o p_{2}\right)\right.$.index ?ops op $1_{1} \neq$ index ?ops op ${ }_{2}$
$\wedge$ are-operators-interfering op op $_{2}$ ) (List.product ?ops ?ops)
let ?fs $=\left[\right.$ encode-interfering-operator-pair-exclusion $\Pi k o p_{1} o p_{2}$.
$\left(o p_{1}, o p_{2}\right) \leftarrow$ ?interfering, $\left.k \leftarrow[0 . .<t]\right]$
have cnf (encode-interfering-operator-exclusion $\Pi t)=\operatorname{cnf}(f o l d r(\wedge)$ ?fs $(\neg \perp))$
unfolding encode-interfering-operator-exclusion-def
by metis
also have $\ldots=\bigcup(c n f$ ' set ? $f s)$
unfolding cnf-foldr-and $[o f ? f s]$..
finally show ?thesis
unfolding cnf-of-encode-interfering-operator-exclusion-is-iii $[$ of $\Pi t]$ by blast
qed
lemma cnf-of-encode-interfering-operator-exclusion-contains-clause-if:
fixes $\Pi$ :: 'variable strips-problem
assumes $k<t$
and op $p_{1} \in$ set (strips-problem.operators-of $\Pi$ ) and $o p_{2} \in$ set (strips-problem.operators-of
П)
and index (strips-problem.operators-of $\Pi$ ) op $p_{1} \neq$ index (strips-problem.operators-of
П) $o p_{2}$
and are-operators-interfering $o p_{1} o p_{2}$
shows $\left\{\left(\text { Operator } k \text { (index (strips-problem.operators-of } \Pi \text { ) op }{ }_{1}\right)\right)^{-1}$
, (Operator $k$ (index (strips-problem.operators-of $\Pi$ ) op $\left.\left.\left.)_{2}\right)\right)^{-1}\right\}$
$\in \operatorname{cnf}$ (encode-interfering-operator-exclusion $\Pi t$ )
proof -
let ?ops $=$ strips-problem.operators-of $\Pi$
and $? \Phi_{X}=$ encode-interfering-operator-exclusion $\Pi t$
let ?Ops $=\left\{\left(o p_{1}, o p_{2}\right) \in \operatorname{set}(\right.$ operators-of $\Pi) \times$ set (operators-of $\left.\Pi\right)$.
index ?ops op $p_{1} \neq$ index ?ops op $2_{2} \wedge$ are-operators-interfering op $\left.{ }_{1} o p_{2}\right\}$
and ? $f=\lambda\left(o p_{1}, o p_{2}\right) .\left\{\left\{\left\{\left(\text { Operator } k(\text { index ?ops op })_{1}\right)\right)^{-1}\right.\right.$, (Operator $k$ (index ?ops op 2$\left.\left.))^{-1}\right\}\right\}$
$\mid k . k \in\{0 . .<t\}\}$
let ? $A=\left(\bigcup\left(o p_{1}, o p_{2}\right) \in\right.$ ?Ops. ?f $\left.\left(o p_{1}, o p_{2}\right)\right)$
let $? B=\bigcup ? A$
and ?C $C=\left\{\left(\text { Operator } k(\text { index ?ops op })_{1}\right)\right)^{-1},\left(\text { Operator } k\left(\text { index ?ops op }{ }_{2}\right)\right)^{-1}$ \} \{
have $\left(o p_{1}, o p_{2}\right) \in ? O p s$
using $\operatorname{assms}(2,3,4,5)$
unfolding operators-of-def
by force
moreover have $\{? C\} \in ? f\left(o p_{1}, o p_{2}\right)$
using assms(1)
by auto
moreover have $\{? C\} \in ? A$
using UN-iff[of ?C - ?Ops] calculation(1, 2)
by blast
ultimately have $\exists X \in ? A . ? C \in X$
by auto


## \}

thus ?thesis
unfolding cnf-of-encode-interfering-operator-exclusion-is
using Union-iff [of ?C ? A]
by auto
qed
lemma is-cnf-encode-interfering-operator-exclusion:
fixes $\Pi$ :: 'variable strips-problem
shows is-cnf (encode-interfering-operator-exclusion $\Pi t$ )
proof -
let ?ops $=$ strips-problem.operators-of $\Pi$
let ?interfering $=$ filter $\left(\lambda\left(o p_{1}, o p_{2}\right)\right.$. index ?ops op $p_{1} \neq$ index ?ops op ${ }_{2}$
$\wedge$ are-operators-interfering op op $_{2}$ ) (List.product ?ops ?ops)
let ?fs $=\left[\right.$ encode-interfering-operator-pair-exclusion $\Pi k o p_{1} o p_{2}$.
$\left(o p_{1}, o p_{2}\right) \leftarrow$ ? interfering, $\left.k \leftarrow[0 . .<t]\right]$
let ? $F s=\left(\bigcup\left(o p_{1}, o p_{2}\right)\right.$
$\in\left\{\left(o p_{1}, o p_{2}\right) \in \operatorname{set}\right.$ (operators-of $\left.\Pi\right) \times \operatorname{set}($ operators-of $\Pi)$.are-operators-interfering
$\left.o p_{1} o p_{2}\right\}$.
( $\lambda k$. encode-interfering-operator-pair-exclusion $\left.\Pi k o p_{1} o p_{2}\right)$ ' $\{0 . .<t\}$ )
\{
fix $f$
assume $f \in$ set? $f_{s}$
then have $f \in$ ?Fs
unfolding cnf-of-encode-interfering-operator-exclusion-is-ii
by blast
then obtain $o p_{1} o p_{2}$
where $\left(o p_{1}, o p_{2}\right) \in \operatorname{set}($ operators-of $\Pi) \times$ set (operators-of $\Pi$ )
and are-operators-interfering op $p_{1} o p_{2}$
and $f \in\left(\lambda k\right.$. encode-interfering-operator-pair-exclusion $\left.\Pi k o p_{1} o p_{2}\right)$ ' $\{0 . .<t\}$
by fast
then obtain $k$ where $f=$ encode-interfering-operator-pair-exclusion $\Pi k o p_{1}$ $o p_{2}$
by blast
then have $f=\neg\left(\right.$ Atom (Operator $k$ (index ?ops op $\left.\left.{ }_{1}\right)\right)$ ) $\vee \neg($ Atom (Operator
$k$ (index ?ops op $\left.{ }_{2}\right)$ ))
unfolding encode-interfering-operator-pair-exclusion-def
by metis
hence $i s-c n f f$
by force
\}
thus ?thesis
unfolding encode-interfering-operator-exclusion-def
using is-cnf-foldr-and-if[of ?fs]
by meson
qed
lemma is-cnf-encode-problem-with-operator-interference-exclusion:
assumes is-valid-problem-strips $\Pi$
shows is-cnf ( $\Phi_{\forall} \Pi t$ )
using is-cnf-encode-problem is-cnf-encode-interfering-operator-exclusion assms
unfolding encode-problem-with-operator-interference-exclusion-def SAT-Plan-Base.encode-problem-def
$i s$-cnf.simps(1)
by blast
lemma cnf-of-encode-problem-with-operator-interference-exclusion-structure:
shows cnf $\left(\Phi_{I} \Pi\right) \subseteq \operatorname{cnf}\left(\Phi_{\forall} \Pi t\right)$
and $c n f\left(\left(\Phi_{G} \Pi\right) t\right) \subseteq c n f\left(\Phi_{\forall} \Pi t\right)$
and $c n f$ (encode-operators $\Pi t) \subseteq c n f\left(\Phi_{\forall} \Pi t\right)$
and cnf (encode-all-frame-axioms $\Pi t) \subseteq \operatorname{cnf}\left(\Phi_{\forall} \Pi t\right)$
and cnf (encode-interfering-operator-exclusion $\Pi t) \subseteq \operatorname{cnf}\left(\Phi_{\forall} \Pi t\right)$
unfolding encode-problem-with-operator-interference-exclusion-def encode-problem-def
SAT-Plan-Base.encode-problem-def
encode-initial-state-def
encode-goal-state-def
by auto+
lemma encode-problem-with-operator-interference-exclusion-has-model-then-also-partial-encodings:
assumes $\mathcal{A} \models \Phi_{\forall} \Pi t$
shows $\mathcal{A} \models$ SAT-Plan-Base.encode-initial-state $\Pi$
and $\mathcal{A} \models$ SAT-Plan-Base.encode-operators $\Pi t$
and $\mathcal{A} \models$ SAT-Plan-Base.encode-all-frame-axioms $\Pi t$
and $\mathcal{A} \models$ encode-interfering-operator-exclusion $\Pi t$
and $\mathcal{A} \models$ SAT-Plan-Base.encode-goal-state $\Pi t$
using assms
unfolding encode-problem-with-operator-interference-exclusion-def encode-problem-def SAT-Plan-Base.encode-problem-def
by $\operatorname{simp}+$
Just as for the basic SATPlan encoding we defined local context for the SATPlan encoding with interfering operator exclusion. We omit this here since it is basically identical to the one shown in the basic SATPlan theory replacing only the definitions of and. The sublocale proof is shown below. It confirms that the new encoding again a CNF as required by locale .

### 8.1 Soundness

The Proof of soundness for the SATPlan encoding with interfering operator exclusion follows directly from the proof of soundness of the basic SATPlan encoding. By looking at the structure of the new encoding which simply extends the basic SATPlan encoding with a conjunct, any model for encoding with exclusion of operator interference also models the basic SATPlan encoding and the soundness of the new encoding therefore follows from theorem ??.
Moreover, since we additionally added interfering operator exclusion clauses at every timestep, the decoded parallel plan cannot contain any interfering operators in any parallel operator (making it serializable).

```
lemma encode-problem-serializable-sound- \(i\) :
    assumes is-valid-problem-strips \(\Pi\)
        and \(\mathcal{A} \models \Phi_{\forall} \Pi t\)
        and \(k<t\)
        and ops \(\in\) set (subseqs \(\left.\left(\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\right)\right)\)
    shows are-all-operators-non-interfering ops
proof -
    let ?ops \(=\) strips-problem.operators-of \(\Pi\)
        and \(? \pi=\Phi^{-1} \Pi \mathcal{A} t\)
        and \(? \Phi_{X}=\) encode-interfering-operator-exclusion \(\Pi t\)
    let \(? \pi_{k}=\left(\Phi^{-1} \Pi \mathcal{A} t\right)!k\)
    \(\{\)
    fix \(C\)
    assume \(C\)-in: \(C \in c n f ? \Phi_{X}\)
    have cnf-semantics \(\mathcal{A}\left(c n f ? \Phi_{X}\right)\)
        using cnf-semantics-monotonous-in-cnf-subsets-if[OF assms(2)
                is-cnf-encode-problem-with-operator-interference-exclusion[OF assms(1)]
                cnf-of-encode-problem-with-operator-interference-exclusion-structure(5)].
    hence clause-semantics \(\mathcal{A} C\)
        unfolding cnf-semantics-def
        using \(C\)-in
        by fast
```

```
} note n\mp@subsup{b}{1}{}=this
{
    fix }o\mp@subsup{p}{1}{}o\mp@subsup{p}{2}{
    assume op 
        and index-op -is-not-index-op 2: index ?ops op 1 }=\mathrm{ index ?ops op 2
    moreover have op -in:op}\mp@code{\mp@code{set ?ops and \mathcal{A}-models-op }
?ops op ())
            and op -in: op 2 }\in\mathrm{ set ?ops and }\mathcal{A}\mathrm{ -models-op 2: A (Operator k (index ?ops
op2))
            using decode-plan-step-element-then[OF assms(3)] calculation
            unfolding decode-plan-def
            by blast+
    moreover {
                            let ?C = {(Operator k (index ?ops op 1) )}\mp@subsup{)}{}{-1},(\mathrm{ Operator k (index ?ops op 2})\mp@subsup{)}{}{-1
}
    assume are-operators-interfering op (1 op 2
    moreover have ?C }\in\operatorname{cnf}?\mp@subsup{\Phi}{X}{
        using cnf-of-encode-interfering-operator-exclusion-contains-clause-if[OF
            assms(3) op 1-in op 2-in index-op 1-is-not-index-op 2] calculation
        by blast
    moreover have \negclause-semantics \mathcal{A ?C}
        using \mathcal{A}}\mp@subsup{\mathrm{ -models-op p}}{1}{}\mathcal{A}\mathrm{ -models-op 
        unfolding clause-semantics-def
        by auto
    ultimately have False
        using n\mp@subsup{b}{1}{}
        by blast
    }
    ultimately have \negare-operators-interfering op ( op 2
    by blast
} note nb 
show ?thesis
    using assms
    proof (induction ops)
    case (Cons op ops)
    have are-all-operators-non-interfering ops
    using Cons.IH[OF Cons.prems(1, 2, 3) Cons-in-subseqsD[OF Cons.prems(4)]]
        by blast
    moreover {
        fix }o\mp@subsup{p}{2}{
        assume op -in-ops:op 
        moreover have o\mp@subsup{p}{1}{}-in-\pi}\mp@subsup{\pi}{k}{}\mathrm{ :op 的 & set ? }\mp@subsup{\pi}{k}{}\mathrm{ and op 
                using element-of-subseqs-then-subset[OF Cons.prems(4)] calculation(1)
                by auto+
            moreover
            {
                have distinct (op % # ops)
                    using subseqs-distinctD[OF Cons.prems(4)]
                    decode-plan-step-distinct[OF Cons.prems(3)]
```

```
            unfolding decode-plan-def
            by blast
            moreover have op 
                using decode-plan-step-element-then(1)[OF Cons.prems(3)] op 1-in-\pi
op2-in-\pi
            unfolding decode-plan-def
            by force+
            moreover have op
                using op 2-in-ops calculation(1)
            by fastforce
            ultimately have index ?ops op 
                    using index-eq-index-conv
            by auto
        }
            ultimately have }\neg\mathrm{ are-operators-interfering op ( op 2
            using nb 
            by blast
        }
        ultimately show ?case
            using list-all-iff
            by auto
    qed simp
qed
theorem encode-problem-serializable-sound:
    assumes is-valid-problem-strips \Pi
        and \mathcal{A}\models\mp@subsup{\Phi}{\forall}{}\Pit
    shows is-parallel-solution-for-problem \Pi ( }\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t
        and}\forallk<length ( (\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t).\mathrm{ are-all-operators-non-interfering (( }\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t
! k)
proof -
    {
        have \mathcal{A }=\mathrm{ SAT-Plan-Base.encode-initial-state П}
            and \mathcal{A}\modelsSAT-Plan-Base.encode-operators \Pit
            and \mathcal{A}\modelsSAT-Plan-Base.encode-all-frame-axioms \Pit
            and \mathcal{A}\modelsSAT-Plan-Base.encode-goal-state \Pit
            using assms(2)
            unfolding encode-problem-with-operator-interference-exclusion-def
            by simp+
    then have \mathcal{A }=SAT-Plan-Base.encode-problem \Pit
            unfolding SAT-Plan-Base.encode-problem-def
            by simp
    }
    thus is-parallel-solution-for-problem \Pi ( }\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t
            using encode-problem-parallel-sound assms(1, 2)
            unfolding decode-plan-def
            by blast
next
    let ? }\pi=\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}
```

```
{
    fix }
    assume k<t
    moreover have ?\pi!k\in set (subseqs (?\pi!k))
        using subseqs-refl
        by blast
    ultimately have are-all-operators-non-interfering (?\pi!k)
        using encode-problem-serializable-sound-i[OF assms]
        unfolding SAT-Plan-Base.decode-plan-def decode-plan-def
        by blast
    }
    moreover have length ? }\pi=
    unfolding SAT-Plan-Base.decode-plan-def decode-plan-def
    by simp
    ultimately show }\forallk<length ?\pi. are-all-operators-non-interfering (? \pi ! k
    by simp
qed
```


### 8.2 Completeness

lemma encode-problem-with-operator-interference-exclusion-complete- $i$ :
assumes is-valid-problem-strips $\Pi$
and is-parallel-solution-for-problem $\Pi \pi$
and $\forall k<$ length $\pi$. are-all-operators-non-interfering $(\pi!k)$
shows valuation-for-plan $\Pi \pi \vDash$ encode-interfering-operator-exclusion $\Pi$ (length
$\pi)$
proof -
let ? $\mathcal{A}=$ valuation-for-plan $\Pi \pi$
and $? \Phi_{X}=$ encode-interfering-operator-exclusion $\Pi$ (length $\left.\pi\right)$
and ?ops $=$ strips-problem.operators-of $\Pi$
and $? t=$ length $\pi$
let ? $\tau=$ trace-parallel-plan-strips $\left((\Pi)_{I}\right) \pi$
let ?Ops $=\left\{\left(o p_{1}, o p_{2}\right) .\left(o p_{1}, o p_{2}\right) \in\right.$ set (operators-of $\left.\Pi\right) \times$ set (operators-of П)
$\wedge$ index ?ops op $1_{1} \neq$ index ?ops op ${ }_{2}$
$\wedge$ are-operators-interfering op $\left.{ }_{1} o p_{2}\right\}$
and ?f $=\lambda\left(o p_{1}, o p_{2}\right)$. $\left\{\left\{\left\{(\text { Operator } k(\text { index ?ops op } 1))^{-1}\right.\right.\right.$, (Operator $k($ index
?ops op 2 $\left.\left.\left.^{2}\right)^{-1}\right\}\right\}$
$\mid k . k \in\{0 . .<$ length $\pi\}\}$
let ? $A=\bigcup$ (?f'? Ops $)$
let $? B=\bigcup ? A$
have $n b_{1}: \forall$ ops $\in$ set $\pi . \forall$ op $\in$ set ops. op $\in$ set (operators-of $\Pi$ )
using is-parallel-solution-for-problem-operator-set[OF assms(2)]
unfolding operators-of-def
by blast

## \{

fix $k o p$
assume $k<$ length $\pi$ and op $\in \operatorname{set}(\pi!k)$


```
    using encode-problem-parallel-complete-vi-a[OF assms(2)]
        encode-problem-parallel-complete-vi-b[OF assms(2)] initial-of-def
    by(cases k< length ? }\tau-1\mathrm{ ; simp)
} note n\mp@subsup{b}{2}{}=this
{
    fix kop,
    assume k< length }
        and op 
        and index ?ops op 1}\not=\mathrm{ index ?ops op 2
    and are-operators-interfering op ( op 2
    moreover have are-all-operators-non-interfering ( }\pi!k
    using assms(3) calculation(1)
    by blast
    moreover have op ( }\not=o\mp@subsup{p}{2}{
    using calculation(3)
    by blast
    ultimately have op, & set ( }\pi!k
    using are-all-operators-non-interfering-set-contains-no-distinct-interfering-operator-pairs
        assms(3)
    by blast
} note nb 
{
    fix C
    assume C\incnf ?'\Phi
    then have C\in?B
        using cnf-of-encode-interfering-operator-exclusion-is[of П length \pi]
        by argo
    then obtain }\mp@subsup{C}{}{\prime}\mathrm{ where }\mp@subsup{C}{}{\prime}\in?A\mathrm{ and C-in:C }\in\mp@subsup{C}{}{\prime
        using Union-iff[of C ?A]
        by meson
    then obtain o\mp@subsup{p}{1}{}o\mp@subsup{p}{2}{}\mathrm{ where (op ,op (op) f set (operators-of П) }\times\mathrm{ set (operators-of}
П)
    and index-op (is-not-index-op 2: index ?ops op 1 }=\mathrm{ index ?ops op 
    and are-operators-interfering-op (-op 2: are-operators-interfering op (op op
    and }\mp@subsup{C}{}{\prime}\mathrm{ -in: C' }\mp@subsup{C}{}{\prime}\in{{{(\mathrm{ Operator }k(\mathrm{ index ?ops op }1)\mp@subsup{)}{}{-1},(Operator k (index ?ops
op}2)\mp@subsup{)}{}{-1}}
            |.k\in{0..<length \pi}}
            using UN-iff[of C' ?f ?Ops]
            by blast
    then obtain }k\mathrm{ where }k\in{0..<\mathrm{ length }\pi
            and C-is:C={(Operator k (index ?ops op 1))}\mp@subsup{)}{}{-1}\mathrm{ , (Operator k (index ?ops
op}2)\mp@subsup{)}{}{-1}
    using C-in C'-in
    by blast
    then have k-lt-length-\pi: }k<\mathrm{ length }
    by simp
    consider (A) op ( \in set ( }\pi!k
    | (B) op 2 \in set ( }\pi!k
```

```
| (C) \negop }\in\operatorname{set}(\pi!k)\vee\nego\mp@subsup{p}{2}{}\in\operatorname{set}(\pi!k
by linarith
hence clause-semantics?. }
proof (cases)
    case }
    moreover have op 2}\not\in\operatorname{set}(\pi!k
    using n\mp@subsup{b}{3}{}}k\mathrm{ k-lt-length-T calculation index-op 1-is-not-index-op ( are-operators-interfering-op (-op 
        by blast
    moreover have }\neg\mathrm{ ? A (Operator k (index ?ops op 2))
        using encode-problem-parallel-complete-vi-d[OF assms(2) k-lt-length-\pi]
            calculation(2)
        by blast
    ultimately show ?thesis
        using C-is
        unfolding clause-semantics-def
        by force
next
    case B
    moreover have op
    using nb }\mp@subsup{3}{3}{}k\mathrm{ -lt-length- }\pi\mathrm{ calculation index-op (-is-not-index-op ( are-operators-interfering-op (-op 2
        by blast
    moreover have \neg?.\mathcal{A (Operator k (index ?ops op 1}))
        using encode-problem-parallel-complete-vi-d[OF assms(2) k-lt-length-\pi]
            calculation(2)
        by blast
    ultimately show ?thesis
        using C-is
        unfolding clause-semantics-def
        by force
next
    case C
    then show ?thesis
        proof (rule disjE)
            assume op }\not\in\operatorname{set}(\pi!k
            then have }\neg\mathrm{ ? A (Operator }k\mathrm{ (index ?ops op }\mp@subsup{)}{1}{})
            using encode-problem-parallel-complete-vi-d[OF assms(2) k-lt-length-\pi]
                by blast
            thus clause-semantics (valuation-for-plan \Pi\pi) C
                using C-is
                unfolding clause-semantics-def
                by force
next
                assume op 2 & set ( }\pi!k
            then have }\neg\mathrm{ ? . A (Operator }k\mathrm{ (index ?ops op 2))
                    using encode-problem-parallel-complete-vi-d[OF assms(2) k-lt-length-\pi]
                    by blast
            thus clause-semantics (valuation-for-plan \Pi\pi) C
                using C-is
                unfolding clause-semantics-def
```

```
                by force
            qed
        qed
    }
    then have cnf-semantics ?\mathcal{A}(cnf ? }\mp@subsup{\Phi}{X}{}
    unfolding cnf-semantics-def..
thus ?thesis
    using cnf-semantics[OF is-nnf-cnf[OF is-cnf-encode-interfering-operator-exclusion]]
    by fast
qed
```

Similar to the soundness proof, we may reuse the previously established facts about the valuation for the completeness proof of the basic SATPlan encoding (??). To make it clearer why this is true we have a look at the form of the clauses for interfering operator pairs $o p_{1}$ and $o p_{2}$ at the same time index $k$ which have the form shown below:
$\left\{(\text { Operator } k(\text { index ops op } 1))^{-1},(\text { Operator } k(\text { index ops op } 2))^{-1}\right\}$
where ops $\equiv \Pi_{\mathcal{O}}$. Now, consider an operator $o p_{1}$ that is contained in the $k$ th plan step $\pi!k$ (symmetrically for $\left.o p_{2}\right)$. Since $\pi$ is a serializable solution, there can be no interference between $o p_{1}$ and $o p_{2}$ at time $k$. Hence $o p_{2}$ cannot be in $\pi!k$ This entails that for $\mathcal{A} \equiv$ valuation-for-plan $\Pi \pi$ it holds that

$$
\mathcal{A} \models \neg \operatorname{Atom}(\text { Operator } k(\text { index ops op } 2))
$$

and $\mathcal{A}$ therefore models the clause.
Furthermore, if neither is present, than $\mathcal{A}$ will evaluate both atoms to false and the clause therefore evaluates to true as well.
It follows from this that each clause in the extension of the SATPlan encoding evaluates to true for $\mathcal{A}$. The other parts of the encoding evaluate to true as per the completeness of the basic SATPlan encoding (theorem ??).

```
theorem encode-problem-serializable-complete:
    assumes is-valid-problem-strips \Pi
        and is-parallel-solution-for-problem \Pi\pi
        and }\forallk<length \pi. are-all-operators-non-interfering ( \pi!k
    shows valuation-for-plan \Pi\pi}\models\mp@subsup{\Phi}{\forall}{}\Pi\mathrm{ (length }\pi\mathrm{ )
proof -
    let ?\mathcal{A}=\mathrm{ valuation-for-plan }\Pi\pi
        and ?!\mp@subsup{\Phi}{X}{}= encode-interfering-operator-exclusion \Pi (length \pi)
    have ?. }\mathcal{A}\modelsSAT-Plan-Base.encode-problem \Pi (length \pi
            using assms(1, 2) encode-problem-parallel-complete
            by auto
    moreover have ?. }\mathcal{A}=\mathrm{ ? }\mp@subsup{\Phi}{X}{
    using encode-problem-with-operator-interference-exclusion-complete-i[OF assms].
    ultimately show ?thesis
```

```
    unfolding encode-problem-with-operator-interference-exclusion-def encode-problem-def
        SAT-Plan-Base.encode-problem-def
    by force
qed
value stop
lemma encode-problem-forall-step-decoded-plan-is-serializable-i:
    assumes is-valid-problem-strips \Pi
        and \mathcal{A}\models\mp@subsup{\Phi}{\forall}{}\Pit
    shows }(\Pi\mp@subsup{)}{G}{}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-serial-plan ((П)
proof -
    let ?G = (\Pi) }\mp@subsup{)}{}{\prime
        and ?I = (\Pi) I
        and ? }\pi=\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}
    let ? }\mp@subsup{\pi}{}{\prime}=\operatorname{concat}(\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t
        and ?\tau = trace-parallel-plan-strips ?I ? }
        and ?}\sigma=map (decode-state-at \Pi\mathcal{A})[0..<Suc (length ? \pi)
    {
        fix }
        assume k-lt-length-\pi: k< length ? }
        moreover have \mathcal{A}\modelsSAT-Plan-Base.encode-problem \Pit
            using assms(2)
            unfolding encode-problem-with-operator-interference-exclusion-def
                encode-problem-def SAT-Plan-Base.encode-problem-def
            by simp
    moreover have length ?\sigma = length ?\tau
            using encode-problem-parallel-correct-vii assms(1) calculation
            unfolding decode-state-at-def decode-plan-def initial-of-def
            by fast
    ultimately have k< length ? \tau - 1 and k<t
            unfolding decode-plan-def SAT-Plan-Base.decode-plan-def
            by force+
} note nb= this
{
    have ?G }\mp@subsup{\subseteq}{m}{}\mathrm{ execute-parallel-plan ?I ? }
            using encode-problem-serializable-sound assms
            unfolding is-parallel-solution-for-problem-def decode-plan-def
                goal-of-def initial-of-def
            by blast
    hence ?G }\mp@subsup{\subseteq}{m}{}\mathrm{ last (trace-parallel-plan-strips ?I ? }\pi\mathrm{ )
            using execute-parallel-plan-reaches-goal-iff-goal-is-last-element-of-trace
            by fast
}
moreover {
    fix }
    assume k-lt-length-\pi: k< length ?\pi
    moreover have k<length ? \tau - 1 and k<t
```

```
        using nb calculation
        by blast+
    moreover have are-all-operators-applicable (?\tau!k)(?\pi!k)
        and are-all-operator-effects-consistent (?\pi!k)
        using trace-parallel-plan-strips-operator-preconditions calculation(2)
        by blast+
    moreover have are-all-operators-non-interfering (?\pi!k)
        using encode-problem-serializable-sound(2)[OF assms(1, 2)] k-lt-length-\pi
        by blast
    ultimately have are-all-operators-applicable (?\tau !k) (?\pi!k)
    and are-all-operator-effects-consistent (?\pi!k)
    and are-all-operators-non-interfering (?\pi!k)
    by blast+
}
ultimately show ?thesis
    using execute-parallel-plan-is-execute-sequential-plan-if assms(1)
    by metis
qed
lemma encode-problem-forall-step-decoded-plan-is-serializable-ii:
    fixes \Pi :: 'variable strips-problem
    shows list-all (\lambdaop. ListMem op (strips-problem.operators-of \Pi))
    (concat ( }\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}t)
proof -
    let ? }\pi=\mp@subsup{\Phi}{}{-1}\Pi\mathcal{A}
    let ? }\mp@subsup{\pi}{}{\prime}=\mathrm{ concat ? }
    {
        have set ?}\mp@subsup{}{}{\prime}\mp@subsup{\pi}{}{\prime}=\bigcup(\mathrm{ set ' ( }\bigcupk<t.{ decode-plan' \Pi\mathcal{A k }))
        unfolding decode-plan-def decode-plan-set-is set-concat
        by auto
        also have ... = \bigcup(\bigcupk<t.{ set (decode-plan' \Pi\mathcal{A}k)})
        by blast
    finally have set ? }\mp@subsup{\pi}{}{\prime}=(\bigcupk<t. set (decode-plan' \Pi\mathcal{A k})
        by blast
    } note nb=this
    {
        fix op
        assume op f set? }\mp@subsup{\pi}{}{\prime
        then obtain k where k<t and op\in set (decode-plan' П\mathcal{A}k)
            using nb
            by blast
    moreover have op\in set (decode-plan \Pi \mathcal{A t!k)}
        using calculation
        unfolding decode-plan-def SAT-Plan-Base.decode-plan-def
        by simp
    ultimately have op \in set (operators-of \Pi)
```

using decode-plan-step-element-then(1)
unfolding operators-of-def decode-plan-def
by blast
\}
thus ?thesis
unfolding list-all-iff ListMem-iff operators-of-def
by blast
qed
Given the soundness and completeness of the SATPlan encoding with interfering operator exclusion $\Phi_{\forall} \Pi t$, we can now conclude this part with showing that for a parallel plan $\pi \equiv \Phi^{-1} \Pi \mathcal{A} t$ that was decoded from a model $\mathcal{A}$ of $\Phi_{\forall} \Pi t$ the serialized plan $\pi^{\prime} \equiv$ concat $\pi$ is a serial solution for П. To this end, we have to show that

- the state reached by serial execution of $\pi^{\prime}$ subsumes $G$, and
- all operators in $\pi^{\prime}$ are operators contained in $\mathcal{O}$.

While the proof of the latter step is rather straight forward, the proof for the former requires a bit more work. We use the previously established theorem on serial and parallel STRIPS equivalence (theorem ??) to show the serializability of $\pi$ and therefore have to show that $G$ is subsumed by the last state of the trace of $\pi^{\prime}$

$$
G \subseteq_{m} \text { last (trace-sequential-plan-strips } I \pi^{\prime} \text { ) }
$$

and moreover that at every step of the parallel plan execution, the parallel operator execution condition as well as non interference are met

$$
\forall k<\text { length } \pi . \text { are-all-operators-non-interfering }(\pi!k)
$$

. ${ }^{12}$ Note that the parallel operator execution condition is implicit in the existence of the parallel trace for $\pi$ with
$G \subseteq_{m}$ last (trace-parallel-plan-strips I $\pi$ )
warranted by the soundness of $\Phi_{\forall} \Pi t$.
theorem serializable-encoding-decoded-plan-is-serializable:
assumes is-valid-problem-strips $\Pi$
and $\mathcal{A} \models \Phi_{\forall} \Pi t$
shows is-serial-solution-for-problem $\Pi$ (concat $\left.\left(\Phi^{-1} \Pi \mathcal{A} t\right)\right)$
using encode-problem-forall-step-decoded-plan-is-serializable-i[OF assms] encode-problem-forall-step-decoded-plan-is-serializable-ii
unfolding is-serial-solution-for-problem-def goal-of-def

[^10]```
        initial-of-def decode-plan-def
```

    by blast
    end
theory SAT-Solve-SAS-Plus
imports SAS-Plus-STRIPS
SAT-Plan-Extensions
begin

## 9 SAT-Solving of SAS+ Problems

```
lemma sas-plus-problem-has-serial-solution-iff-i:
    assumes is-valid-problem-sas-plus \(\Psi\)
        and \(\mathcal{A} \models \Phi_{\forall}(\varphi \Psi) t\)
    shows is-serial-solution-for-problem \(\Psi\left[\varphi_{O}{ }^{-1} \Psi\right.\) op. op \(\leftarrow \operatorname{concat}\left(\Phi^{-1}(\varphi \Psi) \mathcal{A}\right.\)
\(t)\) ]
proof -
    let \(? \Pi=\varphi \Psi\)
        and \(? \pi^{\prime}=\operatorname{concat}\left(\Phi^{-1}(\varphi \Psi) \mathcal{A} t\right)\)
    let ? \(\psi=\left[\varphi_{O}{ }^{-1} \Psi\right.\) op. op \(\leftarrow\) ? \(\left.\pi^{\prime}\right]\)
    \{
        have is-valid-problem-strips ? \(\Pi\)
            using is-valid-problem-sas-plus-then-strips-transformation-too[OF \(\operatorname{assms}(1)]\).
        moreover have STRIPS-Semantics.is-serial-solution-for-problem? \(\Pi_{?} \pi^{\prime}\)
            using calculation serializable-encoding-decoded-plan-is-serializable \([O F\)
                    - assms(2)]
            unfolding decode-plan-def
            by \(\operatorname{simp}\)
        ultimately have SAS-Plus-Semantics.is-serial-solution-for-problem \(\Psi\) ? \(\psi\)
            using assms(1) serial-strips-equivalent-to-serial-sas-plus
            by blast
    \}
    thus ?thesis
        using serial-strips-equivalent-to-serial-sas-plus[OF assms(1)]
        by blast
qed
lemma sas-plus-problem-has-serial-solution-iff-ii:
    assumes is-valid-problem-sas-plus \(\Psi\)
        and is-serial-solution-for-problem \(\Psi \psi\)
        and \(h=\) length \(\psi\)
    shows \(\exists \mathcal{A}\). \(\left(\mathcal{A} \models \Phi_{\forall}(\varphi \Psi) h\right)\)
proof -
    let ? \(\Pi=\varphi \Psi\)
        and \(? \pi=\varphi_{P} \Psi(\) embed \(\psi)\)
    let \(? \mathcal{A}=\) valuation-for-plan ? \(\Pi ? \pi\)
    let \(? t=\) length \(\psi\)
```

```
have nb: length }\psi=\mathrm{ length ? }
    unfolding SAS-Plus-STRIPS.sas-plus-parallel-plan-to-strips-parallel-plan-def
        sasp-op-to-strips-def
        sas-plus-parallel-plan-to-strips-parallel-plan-def
    by (induction \psi; auto)
have is-valid-problem-strips ?\Pi
    using assms(1) is-valid-problem-sas-plus-then-strips-transformation-too
    by blast
moreover have STRIPS-Semantics.is-parallel-solution-for-problem ?\Pi ?\pi
    using execute-serial-plan-sas-plus-is-execute-parallel-plan-sas-plus[OF assms(1,2)]
        strips-equivalent-to-sas-plus[OF assms(1)]
    by blast
moreover {
    fix }
    assume k< length ? }
    moreover obtain ops' where ops'=?\pi!k
        by simp
    moreover have ops' \in set ? }
        using calculation nth-mem
        by blast
    moreover have ? }\pi=[[\mp@subsup{\varphi}{O}{}\Psi\mathrm{ op. op }\leftarrowops].ops \leftarrow embed \psi
        unfolding SAS-Plus-STRIPS.sas-plus-parallel-plan-to-strips-parallel-plan-def
            sasp-op-to-strips-def
                sas-plus-parallel-plan-to-strips-parallel-plan-def
    moreover obtain ops
        where ops'}=[\mp@subsup{\varphi}{O}{}\Psi\mathrm{ I op.op }\leftarrowops
            and ops \in set (embed \psi)
        using calculation(3, 4)
        by auto
    moreover have ops }\in{[op]|op.op\in\operatorname{set}\psi
        using calculation(6) set-of-embed-is
        by blast
    moreover obtain op
        where ops = [op] and op \in set \psi
        using calculation(7)
        by blast
    ultimately have are-all-operators-non-interfering (?\pi!k)
        by fastforce
}
ultimately show ?thesis
    using encode-problem-serializable-complete nb
    by (auto simp: assms(3))
qed
```

To wrap-up our documentation of the Isabelle formalization, we take a look at the central theorem which combines all the previous theorem to show
that SAS+ problems $\Psi$ can be solved using the planning as satisfiability framework.
A solution $\psi$ for the SAS + problem $\Psi$ exists if and only if a model $\mathcal{A}$ and a hypothesized plan length $t$ exist s.t.

$$
\mathcal{A} \models \Phi_{\forall}(\varphi \Psi) t
$$

for the serializable SATPlan encoding of the corresponding STRIPS problem $\Phi_{\forall} \varphi \Psi \quad t$ exist.
theorem sas-plus-problem-has-serial-solution-iff:
assumes is-valid-problem-sas-plus $\Psi$
shows $(\exists \psi$. is-serial-solution-for-problem $\Psi \psi) \longleftrightarrow\left(\exists \mathcal{A} t . \mathcal{A} \models \Phi_{\forall}(\varphi \Psi) t\right)$
using sas-plus-problem-has-serial-solution-iff-i[OF assms]
sas-plus-problem-has-serial-solution-iff-ii[OF assms]
by blast

## 10 Adding Noop actions to the SAS+ problem

Here we add noop actions to the SAS+ problem to enable the SAT formula to be satisfiable if there are plans that are shorter than the given horizons.
definition empty-sasp-action $\equiv$ (SAS-Plus-Representation.sas-plus-operator.precondition-of

$$
=[],
$$

SAS-Plus-Representation.sas-plus-operator.effect-of $=[] D$
lemma sasp-exec-noops: execute-serial-plan-sas-plus s (replicate $n$ empty-sasp-action)

## $=s$

by (induction $n$ arbitrary: )
( auto simp: empty-sasp-action-def STRIPS-Representation.is-operator-applicable-in-def execute-operator-def)

## definition

prob-with-noop $\Pi \equiv$
(SAS-Plus-Representation.sas-plus-problem.variables-of $=$ SAS-Plus-Representation.sas-plus-problem.varia П,

SAS-Plus-Representation.sas-plus-problem.operators-of $=$ empty-sasp-action
\# SAS-Plus-Representation.sas-plus-problem.operators-of $\Pi$,
SAS-Plus-Representation.sas-plus-problem.initial-of $=$ SAS-Plus-Representation.sas-plus-problem.initial-of $\Pi$,

SAS-Plus-Representation.sas-plus-problem.goal-of $=$ SAS-Plus-Representation.sas-plus-problem.goal-of

## $\Pi$,

SAS-Plus-Representation.sas-plus-problem.range-of $=$ SAS-Plus-Representation.sas-plus-problem.range-of П)
lemma sasp-noops-in-noop-problem: set (replicate n empty-sasp-action) $\subseteq$ set (SAS-Plus-Representation.sas-p (prob-with-noop $\Pi$ ))
by (induction $n$ ) (auto simp: prob-with-noop-def)

```
lemma noops-complete:
    SAS-Plus-Semantics.is-serial-solution-for-problem }\Psi\pi
    SAS-Plus-Semantics.is-serial-solution-for-problem (prob-with-noop \Psi) ((replicate
n empty-sasp-action) @ \pi)
    by(induction n)
    (auto simp:SAS-Plus-Semantics.is-serial-solution-for-problem-def insert list.pred-set
                                sasp-exec-noops prob-with-noop-def Let-def empty-sasp-action-def
elem)
definition rem-noops \equiv filter (\lambdaop. op \not= empty-sasp-action)
lemma sasp-filter-empty-action:
    execute-serial-plan-sas-plus s(rem-noops \pis) = execute-serial-plan-sas-plus s \pis
    by (induction }\pis\mathrm{ arbitrary: s)
        (auto simp: empty-sasp-action-def rem-noops-def)
lemma noops-sound:
    SAS-Plus-Semantics.is-serial-solution-for-problem (prob-with-noop \Psi) \pis \Longrightarrow
        SAS-Plus-Semantics.is-serial-solution-for-problem \Psi (rem-noops \pis)
        by(induction }\pis
            (fastforce simp: SAS-Plus-Semantics.is-serial-solution-for-problem-def insert
list.pred-set
                prob-with-noop-def ListMem-iff rem-noops-def
                    sasp-filter-empty-action[unfolded empty-sasp-action-def rem-noops-def]
                        empty-sasp-action-def)+
lemma noops-valid: is-valid-problem-sas-plus }\Psi\Longrightarrow\mathrm{ is-valid-problem-sas-plus (prob-with-noop
\Psi)
    by (auto simp: is-valid-problem-sas-plus-def prob-with-noop-def Let-def
            empty-sasp-action-def is-valid-operator-sas-plus-def list.pred-set)
lemma sas-plus-problem-has-serial-solution-iff-i':
    assumes is-valid-problem-sas-plus \Psi
    and \mathcal{A}\models\mp@subsup{\Phi}{\forall}{}(\varphi(\mathrm{ prob-with-noop }\Psi))t
    shows SAS-Plus-Semantics.is-serial-solution-for-problem \Psi
                (rem-noops
                        (map (\lambdaop. \varphiOO
                            (concat (\Phi}\mp@subsup{\Phi}{}{-1}(\varphi(\mathrm{ prob-with-noop }\Psi))\mathcal{A}t)))
using assms noops-valid
by (force intro!: noops-sound sas-plus-problem-has-serial-solution-iff-i)
lemma sas-plus-problem-has-serial-solution-iff-ii':
assumes is-valid-problem-sas-plus \(\Psi\)
and SAS-Plus-Semantics.is-serial-solution-for-problem \(\Psi \psi\)
and length \(\psi \leq h\)
shows \(\exists \mathcal{A}\). \(\left(\mathcal{A} \models \Phi_{\forall}(\varphi(\right.\) prob-with-noop \(\left.\Psi)) h\right)\)
using assms
by (fastforce
intro!: assms noops-valid noops-complete
```

```
sas-plus-problem-has-serial-solution-iff-ii
    [where }\psi=(\mathrm{ replicate ( }h-\mathrm{ length }\psi)\mathrm{ empty-sasp-action)@ @] )
end
theory AST-SAS-Plus-Equivalence
    imports AI-Planning-Languages-Semantics.SASP-Semantics SAS-Plus-Semantics
List-Index.List-Index
begin
```


## 11 Proving Equivalence of SAS+ representation and Fast-Downward's Multi-Valued Problem Representation

### 11.1 Translating Fast-Downward's represnetation to SAS+

type-synonym nat-sas-plus-problem $=$ (nat, nat) sas-plus-problem
type-synonym nat-sas-plus-operator $=($ nat, nat) sas-plus-operator
type-synonym nat-sas-plus-plan $=($ nat, nat $)$ sas-plus-plan
type-synonym nat-sas-plus-state $=($ nat, nat $)$ state
definition is-standard-effect :: ast-effect $\Rightarrow$ bool where is-standard-effect $\equiv \lambda($ pre, -, -, - $)$. pre $=[]$
definition is-standard-operator :: ast-operator $\Rightarrow$ bool where is-standard-operator $\equiv \lambda(-,-$, effects, -). list-all is-standard-effect effects
fun rem-effect-implicit-pres:: ast-effect $\Rightarrow$ ast-effect where rem-effect-implicit-pres (preconds, $v$, implicit-pre, eff $)=($ preconds, $v$, None, eff $)$
fun rem-implicit-pres :: ast-operator $\Rightarrow$ ast-operator where
rem-implicit-pres (name, preconds, effects, cost) $=$
(name, (implicit-pres effects) @ preconds, map rem-effect-implicit-pres effects, cost)
fun rem-implicit-pres-ops :: ast-problem $\Rightarrow$ ast-problem where rem-implicit-pres-ops (vars, init, goal, ops) $=(v a r s$, init, goal, map rem-implicit-pres ops)
definition consistent-map-lists $x s 1$ xs2 $\equiv(\forall(x 1, x 2) \in$ set $x s 1 . \forall(y 1, y 2) \in$ set $x s 2$. $x 1=y 1 \longrightarrow x 1=y 2$ )
lemma map-add-comm: $(\bigwedge x . x \in \operatorname{dom} m 1 \wedge x \in \operatorname{dom} m 2 \Longrightarrow m 1 x=m 2 x) \Longrightarrow$ $m 1++m 2=m 2++m 1$
by (fastforce simp add: map-add-def split: option.splits)
lemma first-map-add-submap: $(\bigwedge x . x \in \operatorname{dom} m 1 \wedge x \in \operatorname{dom} m 2 \Longrightarrow m 1 x=m 2$

```
x)\Longrightarrow
    m1++m2 \subseteq}\mp@subsup{\}{m}{}x\Longrightarrowm1\mp@subsup{\subseteq}{m}{}
    using map-add-le-mapE map-add-comm
    by force
lemma subsuming-states-map-add:
    (\bigwedgex. x \in dom m1 \cap dom m2 \Longrightarrowm1 x = m2 }x)
    m1++m2 \subseteqm}s\longleftrightarrow(m1\subseteq\mp@subsup{\coprod}{m}{}s\wedgem2\subseteq\mp@subsup{\subseteq}{m}{}s
    by(auto simp: map-add-le-mapI intro: first-map-add-submap map-add-le-mapE)
lemma consistent-map-lists
    \llbracketdistinct (map fst (xs1 @ xs2)); x dom (map-of xs1) \cap dom (map-of xs2)\rrbracket\Longrightarrow
        (map-of xs1) x = (map-of xs2) x
    apply(induction xs1)
    apply (simp-all add: consistent-map-lists-def image-def)
    using map-of-SomeD
    by fastforce
lemma subsuming-states-append:
    distinct (map fst (xs @ ys)) \Longrightarrow
        (map-of (xs @ ys)) \subseteqm}s\longleftrightarrow((map-of ys) \subseteqm s \ (map-of xs) \subseteqm s)
unfolding map-of-append
apply(intro subsuming-states-map-add)
apply (auto simp add: image-def)
by (metis (mono-tags, lifting) IntI empty-iff fst-conv mem-Collect-eq)
definition consistent-pres-op where
consistent-pres-op op \equiv (case op of (name, pres, effs, cost) }
                                    distinct (map fst (pres @ (implicit-pres effs)))
                                    ^consistent-map-lists pres (implicit-pres effs))
definition consistent-pres-op \({ }^{\prime}\) where
consistent-pres-op'op \equiv(case op of (name, pres, effs, cost) =>
                                    consistent-map-lists pres (implicit-pres effs))
lemma consistent-pres-op-then': consistent-pres-op op \Longrightarrow consistent-pres-op' op
    by(auto simp add: consistent-pres-op'-def consistent-pres-op-def)
lemma rem-implicit-pres-ops-valid-states:
        ast-problem.valid-states (rem-implicit-pres-ops prob) = ast-problem.valid-states
prob
    apply(cases prob)
    by(auto simp add: ast-problem.valid-states-def ast-problem.Dom-def
                    ast-problem.numVars-def ast-problem.astDom-def
                        ast-problem.range-of-var-def ast-problem.num Vals-def)
lemma rem-implicit-pres-ops-lookup-op-None:
    ast-problem.lookup-operator (vars, init, goal, ops) name = None \longleftrightarrow
```

ast-problem.lookup-operator (rem-implicit-pres-ops (vars, init, goal, ops)) name $=$ None
by (induction ops) (auto simp: ast-problem.lookup-operator-def ast-problem.ast $\delta$-def)
lemma rem-implicit-pres-ops-lookup-op-Some-1:
ast-problem.lookup-operator (vars, init, goal, ops) name $=$ Some ( $n, p, v p, e) \Longrightarrow$
ast-problem.lookup-operator (rem-implicit-pres-ops (vars, init, goal, ops)) name

$$
=
$$

Some (rem-implicit-pres ( $n, p, v p, e$ ))
by (induction ops) (fastforce simp: ast-problem.lookup-operator-def ast-problem.ast $\delta$-def) +
lemma rem-implicit-pres-ops-lookup-op-Some-1':
ast-problem.lookup-operator prob name $=$ Some $(n, p, v p, e) \Longrightarrow$
ast-problem.lookup-operator (rem-implicit-pres-ops prob) name $=$
Some (rem-implicit-pres ( $n, p, v p, e$ ))
apply(cases prob)
using rem-implicit-pres-ops-lookup-op-Some-1
by $\operatorname{simp}$
lemma implicit-pres-empty: implicit-pres (map rem-effect-implicit-pres effs) $=[]$ by (induction effs) (auto simp: implicit-pres-def)
lemma rem-implicit-pres-ops-lookup-op-Some-2:
ast-problem.lookup-operator (rem-implicit-pres-ops (vars, init, goal, ops)) name
$=$ Some op
$\Longrightarrow \exists o p^{\prime}$. ast-problem.lookup-operator (vars, init, goal, ops) name $=$ Some op ${ }^{\prime}$
$\wedge$

$$
\left(o p=\text { rem-implicit-pres } o p^{\prime}\right)
$$

by (induction ops) (auto simp: ast-problem.lookup-operator-def ast-problem.ast $\delta$-def implicit-pres-empty image-def)
lemma rem-implicit-pres-ops-lookup-op-Some-2':
ast-problem.lookup-operator (rem-implicit-pres-ops prob) name $=$ Some ( $n, p, e, c$ )
$\Longrightarrow \exists{ }^{\prime}{ }^{\prime}$. ast-problem.lookup-operator prob name $=$ Some op ${ }^{\prime} \wedge$
$\left((n, p, e, c)=\right.$ rem-implicit-pres op $\left.{ }^{\prime}\right)$
apply (cases prob)
using rem-implicit-pres-ops-lookup-op-Some-2
by auto
lemma subsuming-states-def ${ }^{\prime}$ :
$s \in$ ast-problem.subsuming-states prob $p s=(s \in$ (ast-problem.valid-states prob)
$\left.\wedge p s \subseteq_{m} s\right)$
by (auto simp add: ast-problem.subsuming-states-def)
lemma rem-implicit-pres-ops-enabled-1:
$\llbracket(\bigwedge o p . o p \in \operatorname{set}($ ast-problem.ast $\delta$ prob $) \Longrightarrow$ consistent-pres-op op $)$; ast-problem.enabled prob name s $\rrbracket$
ast-problem.enabled (rem-implicit-pres-ops prob) name s
by (fastforce simp: ast-problem.enabled-def rem-implicit-pres-ops-valid-states sub-

```
suming-states-def'
            implicit-pres-empty
    intro!: map-add-le-mapI
    dest: rem-implicit-pres-ops-lookup-op-Some-1'
    split: option.splits)+
context ast-problem
begin
lemma lookup-Some-in\delta: lookup-operator }\pi=\mathrm{ Some op בop set ast 
    by(auto simp: find-Some-iff in-set-conv-nth lookup-operator-def)
end
lemma rem-implicit-pres-ops-enabled-2:
    assumes (\bigwedgeop.op }\in\mathrm{ set (ast-problem.ast 
    shows ast-problem.enabled (rem-implicit-pres-ops prob) name s \Longrightarrow
        ast-problem.enabled prob name s
        using assms[OF ast-problem.lookup-Some-in\delta, unfolded consistent-pres-op-def]
        apply(auto simp: subsuming-states-append rem-implicit-pres-ops-valid-states sub-
suming-states-def'
                ast-problem.enabled-def
            dest!: rem-implicit-pres-ops-lookup-op-Some-2'
            split:option.splits)
    using subsuming-states-map-add consistent-map-lists
    apply (metis Map.map-add-comm dom-map-of-conv-image-fst map-add-le-mapE)
    using map-add-le-mapE by blast
lemma rem-implicit-pres-ops-enabled:
    (\bigwedgeop.op \in set (ast-problem.ast\delta prob) \Longrightarrow consistent-pres-op op) \Longrightarrow
        ast-problem.enabled (rem-implicit-pres-ops prob) name s=ast-problem.enabled
prob name s
    using rem-implicit-pres-ops-enabled-1 rem-implicit-pres-ops-enabled-2
    by blast
context ast-problem
begin
lemma std-eff-enabled[simp]:
    is-standard-operator (name, pres, effs, layer) \Longrightarrows\in valid-states \Longrightarrow(filter
(eff-enabled s) effs) = effs
    by (induction effs) (auto simp: is-standard-operator-def is-standard-effect-def
eff-enabled-def subsuming-states-def)
end
lemma is-standard-operator-rem-implicit: is-standard-operator \((n, p, v p, v) \Longrightarrow\) is-standard-operator (rem-implicit-pres ( \(n, p, v p, v\) ))
by (induction vp) (auto simp: is-standard-operator-def is-standard-effect-def)
```

lemma is-standard-operator-rem-implicit-pres-ops:
$\llbracket(\bigwedge$ op. op $\in$ set (ast-problem.ast $\delta(a, b, c, d)) \Longrightarrow$ is-standard-operator op $)$;
op $\in \operatorname{set}$ (ast-problem.ast $\delta$ (rem-implicit-pres-ops $(a, b, c, d))$ )】
$\Longrightarrow$ is-standard-operator op
by (induction d) (fastforce simp add: ast-problem.ast $\delta$-def image-def dest!: is-standard-operator-rem-implicit)
lemma is-standard-operator-rem-implicit-pres-ops':
$\llbracket o p \in$ set (ast-problem.ast $\delta$ (rem-implicit-pres-ops prob));
( $\bigwedge o p . o p \in \operatorname{set}($ ast-problem.ast $\delta$ prob $) \Longrightarrow$ is-standard-operator op)】
$\Longrightarrow$ is-standard-operator op
apply (cases prob)
using is-standard-operator-rem-implicit-pres-ops
by blast
lemma in-rem-implicit-pres- $\delta$ :
$o p \in$ set (ast-problem.ast $\delta$ prob) $\Longrightarrow$
rem-implicit-pres op $\in$ set (ast-problem.ast $\delta$ (rem-implicit-pres-ops prob))
by (auto simp add: ast-problem.ast $\delta$-def)
lemma rem-implicit-pres-ops-execute:
assumes
$(\bigwedge o p . o p \in$ set (ast-problem.ast $\delta$ prob) $\Longrightarrow$ is-standard-operator op) and
$s \in$ ast-problem.valid-states prob
shows ast-problem.execute (rem-implicit-pres-ops prob) name $s=$ ast-problem.execute
prob name $s$
proof-
have $(n, p s, e s, c) \in$ set (ast-problem.ast $\delta$ prob) $\Longrightarrow$
(filter (ast-problem.eff-enabled prob s) es) $=$ es for $n$ ps es $c$
using assms(2)
by (auto simp add: ast-problem.std-eff-enabled dest!: assms(1))
moreover have $(n, p s, e s, c) \in$ set (ast-problem.ast $\delta$ prob) $\Longrightarrow$
(filter (ast-problem.eff-enabled (rem-implicit-pres-ops prob) s) (map rem-effect-implicit-pres
es))

$$
=\text { map rem-effect-implicit-pres es for } n \text { ps es } c
$$

using assms
by (fastforce simp add: ast-problem.std-eff-enabled rem-implicit-pres-ops-valid-states dest!: is-standard-operator-rem-implicit-pres-ops ${ }^{\prime}$ dest: in-rem-implicit-pres- $\delta$ )
moreover have map-of (map $((\lambda(-, x,-, v) .(x, v))$ o rem-effect-implicit-pres) effs)
by (induction effs) auto
ultimately show ?thesis
by (auto simp add: ast-problem.execute-def rem-implicit-pres-ops-lookup-op-Some-1' split: option.splits dest: rem-implicit-pres-ops-lookup-op-Some-2' ast-problem.lookup-Some-in $\delta$ )
qed

```
lemma rem-implicit-pres-ops-path-to:
    wf-ast-problem prob \Longrightarrow
        (\bigwedgeop.op \in set (ast-problem.ast\delta prob) \Longrightarrow consistent-pres-op op) \Longrightarrow
        (\bigwedgeop.op \in set (ast-problem.ast\delta prob) \Longrightarrow is-standard-operator op) \Longrightarrow
        s\in ast-problem.valid-states prob \Longrightarrow
        ast-problem.path-to (rem-implicit-pres-ops prob) s\pis s'=ast-problem.path-to
prob s \pis s'
    by (induction \pis arbitrary: s)
            (auto simp: rem-implicit-pres-ops-execute rem-implicit-pres-ops-enabled
                        ast-problem.path-to.simps wf-ast-problem.execute-preserves-valid)
lemma rem-implicit-pres-ops-astG[simp]: ast-problem.astG (rem-implicit-pres-ops
prob) =
    ast-problem.astG prob
    apply(cases prob)
    by (auto simp add: ast-problem.astG-def)
lemma rem-implicit-pres-ops-goal[simp]: ast-problem.G (rem-implicit-pres-ops prob)
= ast-problem.G prob
    apply(cases prob)
    using rem-implicit-pres-ops-valid-states
    by (auto simp add: ast-problem.G-def ast-problem.astG-def subsuming-states-def')
lemma rem-implicit-pres-ops-astI[simp]:
    ast-problem.astI (rem-implicit-pres-ops prob) = ast-problem.astI prob
    apply(cases prob)
    by (auto simp add: ast-problem.I-def ast-problem.astI-def subsuming-states-def ')
lemma rem-implicit-pres-ops-init[simp]: ast-problem.I (rem-implicit-pres-ops prob)
= ast-problem.I prob
    apply(cases prob)
    by (auto simp add: ast-problem.I-def ast-problem.astI-def)
lemma rem-implicit-pres-ops-valid-plan:
    assumes wf-ast-problem prob
        (\bigwedgeop.op }\in\mathrm{ set (ast-problem.ast j prob) }\Longrightarrow\mathrm{ consistent-pres-op op)
        (\bigwedgeop.op \in set (ast-problem.ast \delta prob) \Longrightarrow is-standard-operator op)
    shows ast-problem.valid-plan (rem-implicit-pres-ops prob) }\pis=ast-problem.valid-plan
prob \pis
    using wf-ast-problem.I-valid[OF assms(1)] rem-implicit-pres-ops-path-to[OF assms]
    by (simp add: ast-problem.valid-plan-def rem-implicit-pres-ops-goal rem-implicit-pres-ops-init)
lemma rem-implicit-pres-ops-numVars[simp]:
    ast-problem.numVars (rem-implicit-pres-ops prob) = ast-problem.numVars prob
    by (cases prob) (simp add:ast-problem.numVars-def ast-problem.astDom-def)
lemma rem-implicit-pres-ops-num Vals[simp]:
    ast-problem.numVals (rem-implicit-pres-ops prob) x = ast-problem.numVals prob
x
```

by (cases prob) (simp add: ast-problem.numVals-def ast-problem.astDom-def)
lemma in-implicit-pres:
$(x, a) \in$ set $($ implicit-pres effs $) \Longrightarrow(\exists$ epres $v$ vp. $($ epres $, x, v p, v) \in$ set effs $\wedge v p=$ Some a)
by (induction effs) (fastforce simp: implicit-pres-def image-def split: if-splits)+
lemma pair4-eqD: $\left(a 1, a 2, a 3, a_{4}\right)=\left(b 1, b 2, b 3, b_{4}\right) \Longrightarrow a 3=b 3$
by $\operatorname{simp}$
lemma rem-implicit-pres-ops-wf-partial-state:
ast-problem.wf-partial-state (rem-implicit-pres-ops prob) $s=$ ast-problem.wf-partial-state prob s
by (auto simp: ast-problem.wf-partial-state-def)
lemma rem-implicit-pres-wf-operator:
assumes consistent-pres-op op
ast-problem.wf-operator prob op
shows
ast-problem.wf-operator (rem-implicit-pres-ops prob) (rem-implicit-pres op)
proof-
obtain name pres effs cost where op: op $=$ (name, pres, effs, cost)
by (cases op)
hence asses: consistent-pres-op (name, pres, effs, cost)
ast-problem.wf-operator prob (name, pres, effs, cost)
using assms
by auto
hence distinct (map fst ((implicit-pres effs) @ pres))
by (simp only: consistent-pres-op-def) auto
moreover have $x<$ ast-problem.numVars (rem-implicit-pres-ops prob)
$v<$ ast-problem.numVals (rem-implicit-pres-ops prob) $x$
if $(x, v) \in \operatorname{set}(($ implicit-pres effs $)$ @ pres) for $x v$
using that asses
by (auto dest!: in-implicit-pres simp: ast-problem.wf-partial-state-def ast-problem.wf-operator-def)
ultimately have ast-problem.wf-partial-state (rem-implicit-pres-ops prob) ((implicit-pres
effs) @ pres)
by (auto simp only: ast-problem.wf-partial-state-def)
moreover have $(\operatorname{map}(\lambda(-, v,-,-) . v)$ effs $)=$ (map $(\lambda(-, v,-,-) . v)($ map rem-effect-implicit-pres effs $))$
by auto
hence distinct (map ( $\lambda(-, v,-,-) . v)$ (map rem-effect-implicit-pres effs))
using assms(2)
by (auto simp only: op ast-problem.wf-operator-def rem-implicit-pres.simps dest!:
pair4-eqD)
moreover have $(\exists v p$. (epres, $x, v p, v) \in$ set effs $) \longleftrightarrow($ epres,$x$, None,$v) \in$ set (map
rem-effect-implicit-pres effs)
for epres $x v$
by force
ultimately show ?thesis
using assms(2)
by (auto simp: op ast-problem.wf-operator-def rem-implicit-pres-ops-wf-partial-state split: prod.splits)
qed
lemma rem-implicit-pres-ops-in $\delta D: o p \in \operatorname{set}$ (ast-problem.ast $\delta$ (rem-implicit-pres-ops prob))
$\Longrightarrow\left(\exists o p^{\prime} . o p^{\prime} \in \operatorname{set}(\right.$ ast-problem.ast $\delta$ prob $) \wedge o p=$ rem-implicit-pres op $\left.{ }^{\prime}\right)$
by (cases prob) (force simp: ast-problem.ast $\delta$-def)
lemma rem-implicit-pres-ops-well-formed:
assumes $(\bigwedge o p . o p \in \operatorname{set}$ (ast-problem.ast $\delta$ prob) $\Longrightarrow$ consistent-pres-op op)
ast-problem.well-formed prob
shows ast-problem.well-formed (rem-implicit-pres-ops prob)
proof-
have map fst (ast-problem.ast $\delta($ rem-implicit-pres-ops prob) $)=$ map fst (ast-problem.ast $\delta$ prob)
by (cases prob) (auto simp: ast-problem.ast $\delta$-def)
thus ?thesis
using assms
by (auto simp add: ast-problem.well-formed-def rem-implicit-pres-ops-wf-partial-state simp del: rem-implicit-pres.simps dest!: rem-implicit-pres-ops-in $\delta D$ intro!: rem-implicit-pres-wf-operator)
qed
definition is-standard-effect ${ }^{\prime}$
:: ast-effect $\Rightarrow$ bool
where $i s$-standard-effect ${ }^{\prime} \equiv \lambda($ pre, -, vpre, -). pre $=[] \wedge$ vpre $=$ None
definition is-standard-operator ${ }^{\prime}$
:: ast-operator $\Rightarrow$ bool
where is-standard-operator ${ }^{\prime} \equiv \lambda(-,-$, effects, -). list-all is-standard-effect' effects
lemma rem-implicit-pres-is-standard-operator':
is-standard-operator $(n, p, e s, c) \Longrightarrow$ is-standard-operator ${ }^{\prime}$ (rem-implicit-pres ( $\left.n, p, e s, c\right)$ )
by (induction es) (auto simp: is-standard-operator'-def is-standard-operator-def is-standard-effect-def
is-standard-effect'-def)
lemma rem-implicit-pres-ops-is-standard-operator':
( $\bigwedge o p . o p \in \operatorname{set}($ ast-problem.ast $\delta(v s, I, G$,ops $)) \Longrightarrow$ is-standard-operator op)
$\pi \in$ set (ast-problem.ast $\delta($ rem-implicit-pres-ops $(v s, I, G$, ops $))$ ) $\Longrightarrow$ is-standard-operator ${ }^{\prime}$
$\pi$
by (cases ops) (auto simp: ast-problem.ast $\delta$-def dest!: rem-implicit-pres-is-standard-operator')
locale abs-ast-prob $=$ wf-ast-problem +

```
    assumes no-cond-effs: \(\forall \pi \in\) set ast \(\delta\). is-standard-operator \({ }^{\prime} \pi\)
context ast-problem
begin
definition abs-ast-variable-section \(=[0 . .<(\) length astDom \()]\)
definition abs-range-map
    :: (nat \(\rightharpoonup\) nat list)
    where abs-range-map \(\equiv\)
    map-of (zip abs-ast-variable-section
        (map ((גvals. [0..<length vals]) o snd o snd)
            astDom))
end
context abs-ast-prob
begin
lemma is-valid-vars-1: astDom \(\neq[] \Longrightarrow\) abs-ast-variable-section \(\neq[]\)
    by (simp add: abs-ast-variable-section-def)
end
lemma upt-eq-Nil-conv'[simp]: \(([]=[i . .<j])=(j=0 \vee j \leq i)\)
    by (induct \(j\) ) simp-all
lemma map-of-zip-map-Some:
        \(v<\) length \(x s\)
        \(\Longrightarrow(\) map-of \((z i p[0 . .<\) length \(x s](\operatorname{map} f x s)) v)=\operatorname{Some}(f(x s!v))\)
    by (induction xs rule: rev-induct) (auto simp add: nth-append map-add-Some-iff)
lemma map-of-zip-Some:
    \(v<\) length \(x s\)
        \(\Longrightarrow(\) map-of \((\) zip \([0 . .<\) length \(x s] x s) v)=\) Some \((x s!v)\)
    by (induction xs rule: rev-induct) (auto simp add: nth-append map-add-Some-iff)
lemma in-set-zip-lengthE:
    \((x, y) \in \operatorname{set}(z i p[0 . .<\) length \(x s] x s) \Longrightarrow(\llbracket x<\) length \(x s ; x s!x=y \rrbracket \Longrightarrow R) \Longrightarrow\)
R
    by (induction xs rule: rev-induct) (auto simp add: nth-append map-add-Some-iff)
context abs-ast-prob
begin
lemma is-valid-vars-2:
    shows list-all ( \(\lambda v\). abs-range-map \(v \neq\) None) abs-ast-variable-section
    by (auto simp add: abs-range-map-def abs-ast-variable-section-def list.pred-set)
end
```

```
context ast-problem
begin
definition abs-ast-initial-state
    :: nat-sas-plus-state
    where abs-ast-initial-state \equiv map-of (zip [0..<length astI] astI)
end
context abs-ast-prob
begin
lemma valid-abs-init-1: abs-ast-initial-state v}\not=N\mathrm{ None }\longleftrightarrowv\in set abs-ast-variable-section
    by (simp add: abs-ast-variable-section-def num Vars-def wf-initial(1) abs-ast-initial-state-def)
lemma abs-range-map-Some:
    shows v\in set abs-ast-variable-section \Longrightarrow
            (abs-range-map v)=Some [0..<length (snd (snd (astDom!v)))]
    by (simp add: numVars-def abs-range-map-def o-def abs-ast-variable-section-def
map-of-zip-map-Some)
lemma in-abs-v-sec-length: v\in set abs-ast-variable-section }\longleftrightarrowv<length astDom
    by (simp add: abs-ast-variable-section-def)
lemma [simp]: v<length astDom \Longrightarrow(abs-ast-initial-state v)=Some (astI!v)
    using wf-initial(1)[simplified numVars-def, symmetric]
    by (auto simp add: map-of-zip-Some abs-ast-initial-state-def split: prod.splits)
lemma [simp]:v<length astDom \LongrightarrowastI!v<length (snd (snd (astDom!v)))
    using wf-initial(1)[simplified numVars-def, symmetric] wf-initial
    by (auto simp add: numVals-def abs-ast-initial-state-def
            split: prod.splits)
lemma [intro!]: v set abs-ast-variable-section \Longrightarrowx<length (snd (snd (astDom
!v))) \Longrightarrow
            x\in set (the (abs-range-map v))
    using abs-range-map-Some
    by (auto simp add:)
lemma [intro!]: x<length astDom \LongrightarrowastI!x<length (snd (snd (astDom!x)))
    using wf-initial[unfolded numVars-def numVals-def]
    by auto
lemma [simp]: abs-ast-initial-state v =Some a \Longrightarrowa<length (snd (snd (astDom
!v)))
    by(auto simp add: abs-ast-initial-state-def
                        wf-initial(1)[unfolded numVars-def numVals-def, symmetric]
        elim!: in-set-zip-lengthE)
```

```
lemma valid-abs-init-2:
    abs-ast-initial-state v}\not=\mathrm{ None }\Longrightarrow(the (abs-ast-initial-state v))\in set (the (abs-range-map
v))
    using valid-abs-init-1
    by auto
end
context ast-problem
begin
definition abs-ast-goal
    :: nat-sas-plus-state
    where abs-ast-goal \equiv map-of astG
end
context abs-ast-prob
begin
lemma [simp]: wf-partial-state s \Longrightarrow (v,a) \in set s\Longrightarrowv\in set abs-ast-variable-section
    by (auto simp add: wf-partial-state-def abs-ast-variable-section-def num Vars-def
        split: prod.splits)
lemma valid-abs-goal-1: abs-ast-goal v}=\mathrm{ None }\Longrightarrowv\in\mathrm{ set abs-ast-variable-section
    using wf-goal
    by (auto simp add: abs-ast-goal-def dest!: map-of-SomeD)
lemma in-abs-rangeI: wf-partial-state s \Longrightarrow(v,a)\in set s \Longrightarrow(a\in set (the
(abs-range-map v)))
    by (auto simp add: abs-range-map-Some wf-partial-state-def numVals-def split:
prod.splits)
lemma valid-abs-goal-2:
    abs-ast-goal v}=\mathrm{ None }\Longrightarrow(\mathrm{ the (abs-ast-goal v)) { set (the (abs-range-map v))
    using wf-goal
    by (auto simp add: map-of-SomeD weak-map-of-SomeI abs-ast-goal-def intro!:
in-abs-rangeI)
end
context ast-problem
begin
definition abs-ast-operator
    :: ast-operator }=>\mathrm{ nat-sas-plus-operator
    where abs-ast-operator }\equiv\lambda(name, preconditions, effects, cost)
        | precondition-of = preconditions,
```

$$
\text { effect-of }=[(v, x) .(-, v,-, x) \leftarrow \text { effects }] D
$$

end
context abs-ast-prob
begin
lemma abs-rangeI: wf-partial-state $s \Longrightarrow(v, a) \in$ set $s \Longrightarrow$ (abs-range-map $v \neq$ None)
by (auto simp add: wf-partial-state-def abs-range-map-def abs-ast-variable-section-def list.pred-set
num Vars-def
split: prod.splits)
lemma abs-valid-operator-1 [intro!]:
wf-operator op $\Longrightarrow$ list-all $(\lambda(v, a)$. ListMem $v$ abs-ast-variable-section $)$
(precondition-of (abs-ast-operator op))
by (cases op; auto simp add: abs-ast-operator-def wf-operator-def list.pred-set ListMem-iff)
lemma wf-operator-preD: wf-operator (name, pres, effs, cost) $\Longrightarrow$ wf-partial-state pres
by (simp add: wf-operator-def)
lemma abs-valid-operator-2[intro!]:
wf-operator op $\Longrightarrow$
list-all $(\lambda(v, a) \cdot(\exists y$.abs-range-map $v=$ Some $y) \wedge$ ListMem a (the (abs-range-map
$v)$ ))
(precondition-of (abs-ast-operator op))
by (cases op,
auto dest!: wf-operator-preD simp: list.pred-set ListMem-iff abs-ast-operator-def intro!: abs-rangeI[simplified not-None-eq] in-abs-rangeI)
lemma wf-operator-effE: wf-operator (name, pres, effs, cost) $\Longrightarrow$ ( $\llbracket$ distinct $(\operatorname{map}(\lambda(-, v,-,-) . v) e f f s)$;
$\bigwedge$ epres $x$ vp v. (epres, $x, v p, v) \in$ set effs $\Longrightarrow$ wf-partial-state epres;
$\bigwedge$ epres $x$ vp v.(epres, $x, v p, v) \in$ set effs $\Longrightarrow x<$ numVars;
^epres x vp v. (epres, $x, v p, v) \in$ set effs $\Longrightarrow v<$ numVals $x$;
\epres x vp v. (epres, $x, v p, v) \in$ set effs $\Longrightarrow$
case vp of None $\Rightarrow$ True $\mid$ Some $v \Rightarrow v<$ numVals $x \rrbracket$
$\Longrightarrow P$ )
$\Longrightarrow P$
unfolding wf-operator-def
by (auto split: prod.splits)
lemma abs-valid-operator-3':
wf-operator (name, pre, eff, cost) $\Longrightarrow$
list-all $(\lambda(v, a)$. ListMem $v$ abs-ast-variable-section) (map $(\lambda(-, v,-, a) .(v, a))$
eff)
by (fastforce simp add: list.pred-set ListMem-iff abs-ast-variable-section-def im-age-def numVars-def elim!: wf-operator-effE split: prod.splits)
lemma abs-valid-operator-3[intro!]:
wf-operator op $\Longrightarrow$
list-all ( $\lambda(v, a)$. ListMem vabs-ast-variable-section) (effect-of (abs-ast-operator $o p)$ )
by (cases op, simp add: abs-ast-operator-def abs-valid-operator-3')
lemma wf-abs-eff: wf-operator (name, pre, eff, cost) $\Longrightarrow$ wf-partial-state (map $(\lambda(-, v,-, a) .(v, a)) e f f)$
by (elim wf-operator-effe, induction eff)
(fastforce simp: wf-partial-state-def image-def o-def split: prod.split-asm)+
lemma abs-valid-operator-4':
wf-operator (name, pre, eff, cost) $\Longrightarrow$
list-all $(\lambda(v, a)$. (abs-range-map $v \neq$ None $) \wedge$ ListMem a (the (abs-range-map
$v))(\operatorname{map}(\lambda(-, v,-, a) .(v, a)) e f f)$
apply(subst list.pred-set ListMem-iff)+
apply (drule wf-abs-eff)
by (metis (mono-tags, lifting) abs-rangeI case-prodI2 in-abs-rangeI)
lemma abs-valid-operator-4 [intro!]:
wf-operator op $\Longrightarrow$
list-all $(\lambda(v, a) .(\exists y$.abs-range-map $v=$ Some $y) \wedge$ ListMem a (the (abs-range-map v)))
(effect-of (abs-ast-operator op))
using abs-valid-operator-4'
by (cases op, simp add: abs-ast-operator-def)
lemma consistent-list-set: wf-partial-state $s \Longrightarrow$
list-all $\left(\lambda(v, a)\right.$. list-all $\left.\left(\lambda\left(v^{\prime}, a^{\prime}\right) . v \neq v^{\prime} \vee a=a^{\prime}\right) s\right) s$
by (auto simp add: list.pred-set wf-partial-state-def eq-key-imp-eq-value split: prod.splits)
lemma abs-valid-operator-5':
wf-operator (name, pre, eff, cost) $\Longrightarrow$
list-all $\left(\lambda(v, a)\right.$. list-all $\left(\lambda\left(v^{\prime}, a^{\prime}\right) . v \neq v^{\prime} \vee a=a^{\prime}\right)$ pre) pre
apply (drule wf-operator-preD)
by (intro consistent-list-set)
lemma abs-valid-operator-5 [intro!]:
wf-operator $o p \Longrightarrow$
list-all $\left(\lambda(v, a)\right.$. list-all $\left(\lambda\left(v^{\prime}, a^{\prime}\right) . v \neq v^{\prime} \vee a=a^{\prime}\right)$ (precondition-of (abs-ast-operator $o p)$ ))
(precondition-of (abs-ast-operator op))
using abs-valid-operator-5'
by (cases op, simp add: abs-ast-operator-def)

```
lemma consistent-list-set-2: distinct (map fst s)\Longrightarrow
    list-all ( }\lambda(v,a).list-all (\lambda(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime}).v\not=\mp@subsup{v}{}{\prime}\veea=\mp@subsup{a}{}{\prime})s)
    by (auto simp add: list.pred-set wf-partial-state-def eq-key-imp-eq-value split:
prod.splits)
lemma abs-valid-operator-6':
    assumes wf-operator (name, pre, eff, cost)
    shows list-all ( }\lambda(v,a).list-all (\lambda(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime}).v\not=\mp@subsup{v}{}{\prime}\veea=\mp@subsup{a}{}{\prime})(\operatorname{map}(\lambda(-,v,-,a)
(v,a)) eff))
    (map (\lambda(-, v, -, a). (v,a)) eff)
proof-
    have *: map fst (map (\lambda(-,v,-,a). (v,a)) eff ) = (map (\lambda(-,v,-,-).v)eff )
        by (induction eff) auto
    show ?thesis
        using assms
        apply(elim wf-operator-effE)
        apply(intro consistent-list-set-2)
        by (subst *)
qed
lemma abs-valid-operator-6 [intro!]:
    wf-operator op \Longrightarrow
        list-all (\lambda(v,a). list-all ( }\lambda(\mp@subsup{v}{}{\prime},\mp@subsup{a}{}{\prime}).v\not=\mp@subsup{v}{}{\prime}\veea=\mp@subsup{a}{}{\prime})\mathrm{ (effect-of (abs-ast-operator
op)))
            (effect-of (abs-ast-operator op))
    using abs-valid-operator-6'
    by (cases op, simp add: abs-ast-operator-def)
end
context ast-problem
begin
definition abs-ast-operator-section
    :: nat-sas-plus-operator list
    where abs-ast-operator-section }\equiv[abs-ast-operator op.op \leftarrowast\delta
definition abs-prob :: nat-sas-plus-problem
    where abs-prob = 0
    variables-of = abs-ast-variable-section,
    operators-of =abs-ast-operator-section,
    initial-of = abs-ast-initial-state,
    goal-of = abs-ast-goal,
    range-of =abs-range-map
    D
end
```

context abs-ast-prob
begin
lemma $[$ simp $]: o p \in$ set ast $\delta \Longrightarrow$ (is-valid-operator-sas-plus abs-prob) (abs-ast-operator op)
apply (cases op)
apply(subst is-valid-operator-sas-plus-def Let-def)+
using wf-operators(2)
by (fastforce simp add: abs-prob-def)+
lemma abs-ast-operator-section-valid:
list-all (is-valid-operator-sas-plus abs-prob) abs-ast-operator-section
by (auto simp: abs-ast-operator-section-def list.pred-set)
lemma abs-prob-valid: is-valid-problem-sas-plus abs-prob
using valid-abs-goal-1 valid-abs-goal-2 valid-abs-init-1 is-valid-vars-2
abs-ast-operator-section-valid[unfolded abs-prob-def]
by (auto simp add: is-valid-problem-sas-plus-def Let-def ListMem-iff abs-prob-def)
definition abs-ast-plan
:: SASP-Semantics.plan $\Rightarrow$ nat-sas-plus-plan
where abs-ast-plan $\pi s$
$\equiv$ map (abs-ast-operator o the o lookup-operator) $\pi s$
lemma std-then-implici-effs[simp]: is-standard-operator' (name, pres, effs, layer)
$\Longrightarrow$ implicit-pres effs $=$ []
apply(induction effs)
by (auto simp add: is-standard-operator'-def implicit-pres-def is-standard-effect'-def)
lemma [simp]: enabled $\pi s \Longrightarrow$ lookup-operator $\pi=$ Some (name, pres, effs, layer)
$\Longrightarrow$
is-standard-operator' ${ }^{\prime}$ name, pres, effs, layer $) \Longrightarrow$ $($ filter $($ eff-enabled $s)$ effs $)=$ effs
$\mathbf{b y}($ auto simp add: enabled-def is-standard-operator'-def eff-enabled-def is-standard-effect'-def filter-id-conv list.pred-set)
lemma effs-eq-abs-effs: (effect-of (abs-ast-operator (name, pres, effs, layer))) = (map $(\lambda(-, x,-, v) .(x, v))$ effs $)$
by (auto simp add: abs-ast-operator-def split: option.splits prod.splits)
lemma exect-eq-abs-execute:
«enabled $\pi s$; lookup-operator $\pi=$ Some (name, preconds, effs, layer);
is-standard-operator'(name, preconds, effs, layer) $\Longrightarrow$
execute $\pi s=($ execute-operator-sas-plus $s(($ abs-ast-operator o the o lookup-operator $)$
$\pi)$ )
using effs-eq-abs-effs
by (auto simp add: execute-def execute-operator-sas-plus-def)

```
lemma enabled-then-sas-applicable:
    enabled \pis\LongrightarrowSAS-Plus-Representation.is-operator-applicable-in s ((abs-ast-operator
o the o lookup-operator) \pi)
    by (auto simp add: subsuming-states-def enabled-def lookup-operator-def
                SAS-Plus-Representation.is-operator-applicable-in-def abs-ast-operator-def
                split: option.splits prod.splits)
lemma path-to-then-exec-serial: }\forall\pi\in\mathrm{ set }\pi\mathrm{ s. lookup-operator }\pi\not=None 
        path-to s \pis s' \Longrightarrow
        s'\subseteqm}\mp@subsup{\subseteq}{m}{}\mathrm{ execute-serial-plan-sas-plus s (abs-ast-plan }\pis
proof(induction \pis arbitrary:s s')
    case (Cons a }\pis\mathrm{ )
    then show ?case
    by (force simp: exect-eq-abs-execute abs-ast-plan-def lookup-Some-in\delta no-cond-effs
                dest: enabled-then-sas-applicable)
qed (auto simp: execute-serial-plan-sas-plus-def abs-ast-plan-def)
lemma map-of-eq-None-iff:
    (None = map-of xys x)=(x\not\infst'(set xys))
by (induct xys) simp-all
lemma [simp]: I = abs-ast-initial-state
    apply(intro HOL.ext)
    by (auto simp: map-of-eq-None-iff set-map[symmetric] I-def abs-ast-initial-state-def
map-of-zip-Some
        dest: map-of-SomeD)
lemma [simp]: }\forall\pi\in\mathrm{ set }\pi\mathrm{ s. lookup-operator }\pi\not=\mathrm{ None }
                op\inset (abs-ast-plan }\pis)\Longrightarrowop\in\mathrm{ set abs-ast-operator-section
    by (induction \pis) (auto simp: abs-ast-plan-def abs-ast-operator-section-def lookup-Some-in\delta)
end
context ast-problem
begin
lemma path-to-then-lookup-Some:(\exists\mp@subsup{s}{}{\prime}\inG. path-to s \pis s})\Longrightarrow(\forall\pi\in\mathrm{ set }\pis\mathrm{ .
lookup-operator }\pi\not=\mathrm{ None)
    by (induction \pis arbitrary: s) (force simp add: enabled-def split: option.splits)+
lemma valid-plan-then-lookup-Some: valid-plan }\pis\Longrightarrow(\forall\pi\in\mathrm{ set }\pi\mathrm{ s. lookup-operator
\pi}\not=\mathrm{ None)
    using path-to-then-lookup-Some
    by(simp add: valid-plan-def)
end
context abs-ast-prob
```


## begin

```
theorem valid-plan-then-is-serial-sol:
    assumes valid-plan }\pi
    shows is-serial-solution-for-problem abs-prob (abs-ast-plan \pis)
    using valid-plan-then-lookup-Some[OF assms] assms
    by (auto simp add: is-serial-solution-for-problem-def valid-plan-def initial-of-def
                    abs-prob-def abs-ast-goal-def G-def subsuming-states-def list-all-iff
                        ListMem-iff map-le-trans path-to-then-exec-serial
        simp del: sas-plus-problem.select-defs)
```

end

### 11.2 Translating SAS+ represnetation to Fast-Downward's

 context ast-problembegin
definition lookup-action:: nat-sas-plus-operator $\Rightarrow$ ast-operator option where
lookup-action op $\equiv$
find ( $\lambda(-$, pres, effs, -$)$. precondition-of op $=$ pres $\wedge$
$\operatorname{map}(\lambda(v, a) .([], v$, None, $a))($ effect-of op $)=$ effs $)$
ast $\delta$
end
context abs-ast-prob
begin
lemma find-Some: find $P x s=$ Some $x \Longrightarrow x \in$ set $x s \wedge P x$
by (auto simp add: find-Some-iff)
lemma distinct-find: distinct (map $f x s) \Longrightarrow x \in \operatorname{set} x s \Longrightarrow$ find $\left(\lambda x^{\prime} . f x^{\prime}=f x\right)$
$x s=$ Some $x$
by (induction xs) (auto simp: image-def)
lemma lookup-operator-find: lookup-operator nme $=$ find ( $\lambda$ op. $f s t$ op $=n m e$ ) ast $\delta$
by (auto simp: lookup-operator-def intro!: arg-cong[where $f=(\lambda x$. find $x$ ast $\delta)])$
lemma lookup-operator-works-1: lookup-action op $=$ Some $\pi^{\prime} \Longrightarrow$ lookup-operator
$\left(\right.$ fst $\left.\pi^{\prime}\right)=$ Some $\pi^{\prime}$
by (auto simp: wf-operators(1) lookup-operator-find lookup-action-def dest: find-Some
intro: distinct-find)
lemma lookup-operator-works-2:
lookup-action (abs-ast-operator (name, pres, effs, layer)) = Some (name ${ }^{\prime}$, pres ${ }^{\prime}$,
effs', layer')
$\Longrightarrow$ pres $=$ pres $^{\prime}$
by (auto simp: lookup-action-def abs-ast-operator-def dest!: find-Some)

```
lemma [simp]: is-standard-operator' (name, pres, effs, layer) \Longrightarrow
```

            map \((\lambda(v, a) .([], v\), None, a) ) (effect-of (abs-ast-operator (name, pres, effs,
    layer $)$ )) $=$ effs
by (induction effs) (auto simp: is-standard-operator'-def abs-ast-operator-def
is-standard-effect'-def)
lemma lookup-operator-works-3:
is-standard-operator' ${ }^{\prime}$ name, pres, effs, layer $) \Longrightarrow$ (name, pres, effs, layer $) \in$ set ast $\delta \Longrightarrow$
lookup-action (abs-ast-operator (name, pres, effs, layer)) = Some (name', pres', effs', layer') $\Longrightarrow$ effs $=$ effs ${ }^{\prime}$
by (auto simp: is-standard-operator'-def lookup-action-def dest!: find-Some)
lemma mem-find-Some: $x \in$ set $x s \Longrightarrow P x \Longrightarrow \exists x^{\prime}$. find $P$ xs $=$ Some $x^{\prime}$ by (induction xs) auto
lemma [simp]: precondition-of (abs-ast-operator $(x 1, a, a a, b))=a$ by (simp add: abs-ast-operator-def)
lemma std-lookup-action: is-standard-operator' ${ }^{\prime}$ ast-op $\Longrightarrow$ ast-op $\in$ set ast $\delta \Longrightarrow$
$\exists$ ast-op' ${ }^{\prime}$ lookup-action (abs-ast-operator ast-op) $=$ Some

## ast-op ${ }^{\prime}$

unfolding lookup-action-def
apply (intro mem-find-Some)
by (auto split: prod.splits simp: o-def)
lemma is-applicable-then-enabled-1:
ast-op $\in$ set ast $\delta \Longrightarrow$
$\exists$ ast-op'. lookup-operator ( (fst o the o lookup-action o abs-ast-operator) ast-op)
$=$ Some ast-op ${ }^{\prime}$
using lookup-operator-works-1 std-lookup-action no-cond-effs
by auto
lemma lookup-action-Some-in- $\delta$ : lookup-action op $=$ Some ast-op $\Longrightarrow$ ast-op $\in$ set ast $\delta$
using lookup-operator-works-1 lookup-Some-ind by fastforce
lemma lookup-operator-eq-name: lookup-operator name $=$ Some (name ${ }^{\prime}$, pres, effs, layer $) \Longrightarrow$ name $=$ name ${ }^{\prime}$
using lookup-operator-wf(2)
by fastforce
lemma eq-name-eq-pres: (name, pres, effs, layer) $\in$ set ast $\delta \Longrightarrow$ (name, pres', effs $s^{\prime}$, layer $\left.{ }^{\prime}\right) \in$ set ast $\delta$
$\Longrightarrow$ pres $=$ pres ${ }^{\prime}$
using eq-key-imp-eq-value[OF wf-operators(1)]
by auto
lemma eq－name－eq－effs：

```
    name \(=\) name \({ }^{\prime} \Longrightarrow(\) name, pres, effs, layer \() \in\) set ast \(\delta \Longrightarrow\left(\right.\) name \(^{\prime}\), pres \(^{\prime}\), effs \({ }^{\prime}\),
```

layer $\left.{ }^{\prime}\right) \in$ set ast $\delta$
$\Longrightarrow$ effs $=$ effs $s^{\prime}$
using eq-key-imp-eq-value[OF wf-operators(1)]
by auto
lemma is-applicable-then-subsumes:
$s \in$ valid-states $\Longrightarrow$
SAS-Plus-Representation.is-operator-applicable-in s (abs-ast-operator (name,
pres, effs, layer)) $\Longrightarrow$
$s \in$ subsuming-states (map-of pres)
by (simp add: subsuming-states-def SAS-Plus-Representation.is-operator-applicable-in-def
abs-ast-operator-def)
lemma eq-name-eq-pres':
$\llbracket s \in$ valid-states ; is-standard-operator' ${ }^{\prime}$ (name, pres, effs, layer); (name, pres,
effs, layer) $\in$ set ast $\delta$;
lookup-operator ((fst o the o lookup-action o abs-ast-operator) (name, pres, effs,
layer $)$ ) $=$ Some (name', pres', effs', layer $\left.{ }^{\prime}\right) 】$
$\Longrightarrow$ pres $=$ pres ${ }^{\prime}$
using lookup-operator-eq-name lookup-operator-works-2
by (fastforce dest!: std-lookup-action
simp: eq-name-eq-pres[OF lookup-action-Some-in-ס lookup-Some-in $\delta]$ )
lemma is-applicable-then-enabled-2:
$\llbracket s \in$ valid-states $;$ ast-op $\in$ set ast $\delta$;
SAS-Plus-Representation.is-operator-applicable-in s (abs-ast-operator ast-op);
lookup-operator ((fst o the o lookup-action o abs-ast-operator) ast-op $)=$ Some
( name, pres, effs, layer)】
$\Longrightarrow s \in$ subsuming-states (map-of pres)
apply (cases ast-op)
using eq-name-eq-pres' is-applicable-then-subsumes no-cond-effs
by fastforce
lemma is-applicable-then-enabled-3:
$\llbracket s \in$ valid-states;
lookup-operator ((fst o the o lookup-action o abs-ast-operator) ast-op $)=$ Some
( name, pres, effs, layer)】
$\Longrightarrow s \in$ subsuming-states (map-of (implicit-pres effs))
apply (cases ast-op)
using no-cond-effs
by (auto dest!: std-then-implici-effs std-lookup-action lookup-Some-inס
simp: subsuming-states-def)
lemma is-applicable-then-enabled:
$\llbracket s \in$ valid-states; ast-op $\in$ set ast $\delta ;$
SAS-Plus-Representation.is-operator-applicable-in s (abs-ast-operator ast-op)】
$\Longrightarrow$ enabled ( $(f s t$ o the o lookup-action o abs-ast-operator) ast-op) s using is-applicable-then-enabled-1 is-applicable-then-enabled-2 is-applicable-then-enabled-3 by (simp add: enabled-def split: option.splits)
lemma eq-name-eq-effs':
assumes lookup-operator ((fst o the o lookup-action o abs-ast-operator) (name,
pres, effs, layer)) $=$
Some (name', pres', effs', layer')
is-standard-operator' (name, pres, effs, layer) (name, pres, effs, layer) $\in$
set ast $\delta$
$s \in$ valid-states
shows effs $=$ effs ${ }^{\prime}$
using std-lookup-action[OF assms(2,3)] assms
by (auto simp: lookup-operator-works-3[OF assms(2,3)]
eq-name-eq-effs $[$ OF lookup-operator-eq-name lookup-action-Some-in- $\delta$
lookup-Some-in $]$ )
lemma std-eff-enabled' $[$ simp $]$ :
is-standard-operator ${ }^{\prime}$ (name, pres, effs, layer $) \Longrightarrow s \in$ valid-states $\Longrightarrow$ (filter
(eff-enabled s) effs $)=$ effs
by (induction effs) (auto simp: is-standard-operator'-def is-standard-effect'-def
eff-enabled-def subsuming-states-def)

## lemma execute-abs:

$\llbracket s \in$ valid-states $;$ ast-op $\in$ set ast $\delta ;$
SAS-Plus-Representation.is-operator-applicable-in s (abs-ast-operator ast-op)】
$\Longrightarrow$
execute ((fst o the o lookup-action o abs-ast-operator) ast-op) $s=$
execute-operator-sas-plus s (abs-ast-operator ast-op)
using no-cond-effs
by (cases ast-op)
(fastforce simp add: execute-def execute-operator-sas-plus-def effs-eq-abs-effs
dest: is-applicable-then-enabled-1 eq-name-eq-effs ${ }^{\prime}$ [unfolded o-def]
split: option.splits)+
fun sat-preconds-as where
sat-preconds-as $s[]=$ True
| sat-preconds-as s (op\#ops) $=$
(SAS-Plus-Representation.is-operator-applicable-in s op $\wedge$ sat-preconds-as (execute-operator-sas-plus sop) ops)
lemma exec-serial-then-path-to':
$\llbracket s \in$ valid-states;
$\forall o p \in$ set ops. $\exists$ ast-op $\in$ set ast $\delta . o p=a b s$-ast-operator ast-op;
(sat-preconds-as s ops) $\Longrightarrow$
path-to $s$ (map (fst o the o lookup-action) ops) (execute-serial-plan-sas-plus s
ops)
proof(induction ops arbitrary: s)
case (Cons a ops)

```
then show ?case
    using execute-abs is-applicable-then-enabled execute-preserves-valid
    apply simp
    by metis
qed auto
end
fun rem-condless-ops where
    rem-condless-ops s [] = []
| rem-condless-ops s (op#ops) =
    (if SAS-Plus-Representation.is-operator-applicable-in s op then
    op # (rem-condless-ops (execute-operator-sas-plus s op) ops)
    else [])
context abs-ast-prob
begin
lemma exec-rem-consdless: execute-serial-plan-sas-plus s (rem-condless-ops s ops)
= execute-serial-plan-sas-plus s ops
    by (induction ops arbitrary: s) auto
lemma rem-conless-sat: sat-preconds-as s (rem-condless-ops s ops)
    by (induction ops arbitrary: s) auto
lemma set-rem-condlessD: x\in set (rem-condless-ops s ops) \Longrightarrowx\in set ops
    by (induction ops arbitrary: s) auto
lemma exec-serial-then-path-to:
    |s\in valid-states;
    \forallop set ops. \existsast-op\in set ast \delta. op = abs-ast-operator ast-op\rrbracket\Longrightarrow
    path-to s (((map (fst o the o lookup-action)) o rem-condless-ops s) ops)
            (execute-serial-plan-sas-plus s ops)
    using rem-conless-sat
    by (fastforce dest!: set-rem-condlessD
                intro!: exec-serial-then-path-to'
                                    [where s=s and ops = rem-condless-ops s ops,
                                    unfolded exec-rem-consdless])
lemma is-serial-solution-then-abstracted: is-serial-solution-for-problem abs-prob ops
    #op\inset ops. \existsast-op\in set ast\delta.op = abs-ast-operator ast-op
    by(auto simp: is-serial-solution-for-problem-def abs-prob-def Let-def list.pred-set
            ListMem-iff abs-ast-operator-section-def
        split: if-splits)
lemma lookup-operator-works-1': lookup-action op \(=\) Some \(\pi^{\prime} \Longrightarrow \exists\) op. lookup-operator \(\left(f s t \pi^{\prime}\right)=o p\)
using lookup-operator-works-1 by auto
```

```
lemma is-serial-sol-then-valid-plan-1:
    \llbracketis-serial-solution-for-problem abs-prob ops;
        \pi}\mathrm{ set ((map (fst o the o lookup-action) o rem-condless-ops I) ops)】】
        lookup-operator }\pi\not=\mathrm{ None
    using std-lookup-action lookup-operator-works-1 no-cond-effs
    by (fastforce dest!: set-rem-condlessD is-serial-solution-then-abstracted
                simp: valid-plan-def list.pred-set ListMem-iff)
lemma is-serial-sol-then-valid-plan-2:
    \llbracketis-serial-solution-for-problem abs-prob ops\rrbracket\Longrightarrow
    ( \exists}\mp@subsup{s}{}{\prime}\inG. path-to I ((map (fst o the o lookup-action) o rem-condless-ops I) ops
s')
    using I-valid
    by (fastforce intro: path-to-pres-valid exec-serial-then-path-to
        intro!: bexI[where x = execute-serial-plan-sas-plus I ops]
        dest: is-serial-solution-then-abstracted
        simp: list.pred-set ListMem-iff abs-ast-operator-section-def
            G-def subsuming-states-def is-serial-solution-for-problem-def
            abs-prob-def abs-ast-goal-def)+
end
context ast-problem
begin
definition decode-abs-plan \equiv(map (fst o the o lookup-action) o rem-condless-ops
I)
end
context abs-ast-prob
begin
theorem is-serial-sol-then-valid-plan:
    \llbracketis-serial-solution-for-problem abs-prob ops\rrbracket\Longrightarrow
        valid-plan (decode-abs-plan ops)
    using is-serial-sol-then-valid-plan-1 is-serial-sol-then-valid-plan-2
    by(simp add: valid-plan-def decode-abs-plan-def)
end
end
```


### 11.3 SAT encoding works for Fast-Downward's representation

## context abs-ast-prob

begin
theorem is-serial-sol-then-valid-plan-encoded:
$\mathcal{A} \vDash \Phi_{\forall}(\varphi$ (prob-with-noop abs-prob) $) t \Longrightarrow$ valid-plan
(decode-abs-plan
(rem-noops
$\left(\right.$ map $\left(\lambda o p . \varphi_{O}^{-1}(\right.$ prob-with-noop abs-prob) op $)$
$\quad\left(\right.$ concat $\left(\Phi^{-1}(\varphi(\right.$ prob-with-noop abs-prob $\left.\left.\left.\left.)) \mathcal{A} t\right)\right)\right)\right)$
by (fastforce intro!: is-serial-sol-then-valid-plan abs-prob-valid sas-plus-problem-has-serial-solution-iff-i')
lemma length-abs-ast-plan: length $\pi s=$ length (abs-ast-plan $\pi s)$
by (auto simp: abs-ast-plan-def)
theorem valid-plan-then-is-serial-sol-encoded:
valid-plan $\pi s \Longrightarrow$ length $\pi s \leq h \Longrightarrow \exists \mathcal{A} . \mathcal{A} \models \Phi_{\forall}(\varphi($ prob-with-noop abs-prob $))$
$h$
apply(subst (asm) length-abs-ast-plan)
by (fastforce intro!: sas-plus-problem-has-serial-solution-iff-ii' abs-prob-valid valid-plan-then-is-serial-sol)
end

## 12 DIMACS-like semantics for CNF formulae

We now push the SAT encoding towards a lower-level representation by replacing the atoms which have variable IDs and time steps into natural numbers.
lemma gtD: $((l:: n a t)<n) \Longrightarrow(\exists m . n=$ Suc $m \wedge l \leq m)$
by (induction $n$ ) auto
locale cnf-to-dimacs $=$
fixes $h::$ nat and n-ops :: nat
begin
fun var-to-dimacs where
var-to-dimacs (Operator $t k)=1+t+k * h$
| var-to-dimacs $($ State $t k)=1+n$-ops $* h+t+k *(h)$

```
definition dimacs-to-var where
    dimacs-to-var \(v \equiv\)
        if \(v<1+n\)-ops \(* h\) then
            Operator \(((v-1) \bmod (h))((v-1) \operatorname{div}(h))\)
        else
        \((\) let \(k=((v-1)-n\)-ops \(* h)\) in
            State \((k \bmod (h))(k \operatorname{div}(h)))\)
fun valid-state-var where
    valid-state-var (Operator \(t k) \longleftrightarrow t<h \wedge k<n\)-ops
| valid-state-var (State \(t k) \longleftrightarrow t<h\)
lemma State-works:
valid-state-var (State \(t k) \Longrightarrow\)
    dimacs-to-var (var-to-dimacs \((\) State \(t k))=\)
    (State \(t k\) )
    by (induction \(k\) ) (auto simp add: dimacs-to-var-def add.left-commute Let-def)
lemma Operator-works:
    valid-state-var (Operator \(t k) \Longrightarrow\)
        dimacs-to-var (var-to-dimacs \((\) Operator \(t k))=\)
            (Operator \(t k\) )
    by (induction \(k\) ) (auto simp add: algebra-simps dimacs-to-var-def gr0-conv-Suc
nat-le-iff-add dest!: gtD)
lemma sat-plan-to-dimacs-works:
    valid-state-var \(s v \Longrightarrow\)
        dimacs-to-var (var-to-dimacs sv) \(=s v\)
    apply(cases sv)
    using State-works Operator-works
    by auto
end
lemma changing-atoms-works:
    \((\bigwedge x . P x \Longrightarrow(f \circ g) x=x) \Longrightarrow(\forall x \in\) atoms phi. \(P x) \Longrightarrow M \models p h i \longleftrightarrow M o f\)
\(\vDash\) map-formula \(g\) phi
    by (induction phi) auto
lemma changing-atoms-works':
    \(M\) o \(g \models p h i \longleftrightarrow M \models\) map-formula \(g\) phi
    by (induction phi) auto
context cnf-to-dimacs
begin
lemma sat-plan-to-dimacs:
    ( \(\bigwedge\) sv. sv \(\in\) atoms sat-plan-formula \(\Longrightarrow\) valid-state-var sv) \(\Longrightarrow\)
        \(M \models\) sat-plan-formula
```

$\longleftrightarrow M$ o dimacs-to-var $\models$ map-formula var-to-dimacs sat-plan-formula
by (auto intro!: changing-atoms-works[where $P=$ valid-state-var $]$ simp: sat-plan-to-dimacs-works)

## lemma dimacs-to-sat-plan:

M o var-to-dimacs $\models$ sat-plan-formula
$\longleftrightarrow M \models$ map-formula var-to-dimacs sat-plan-formula
using changing-atoms-works ${ }^{\prime}$.
end
locale sat-solve-sasp $=$ abs-ast-prob $\Pi+$ cnf-to-dimacs Suc $h$ Suc (length ast $\delta$ )
for $\Pi h$
begin
lemma encode-initial-state-valid:
$s v \in$ atoms (encode-initial-state Prob) $\Longrightarrow$ valid-state-var sv
by (auto simp add: encode-state-variable-def Let-def encode-initial-state-def split:
sat-plan-variable.splits bool.splits)
lemma length-operators: length (operators-of $(\varphi($ prob-with-noop abs-prob $)))=$ Suc
(length ast $\delta$ )
by (simp add: abs-prob-def abs-ast-operator-section-def sas-plus-problem-to-strips-problem-def
prob-with-noop-def)
lemma encode-operator-effect-valid-1: $t<h \Longrightarrow o p \in$ set (operators-of ( $\varphi$ (prob-with-noop abs-prob))) $\Longrightarrow$
$s v \in$ atoms
( $\bigwedge$ (map $(\lambda v$.
$\neg($ Atom $($ Operator $t($ index (operators-of $(\varphi($ prob-with-noop abs-prob $)))$
$o p)$ )

$$
\begin{aligned}
& \vee \text { Atom }(\text { State }(\text { Suc } t)(\text { index vs } v))) \\
& \text { asses })) \Longrightarrow \\
& \text { valid-state-var sv }
\end{aligned}
$$

using length-operators
by (induction asses) (auto simp: simp add: cnf-to-dimacs.valid-state-var.simps)
lemma encode-operator-effect-valid-2: $t<h \Longrightarrow o p \in$ set (operators-of ( $\varphi$ (prob-with-noop
abs-prob))) $\Longrightarrow$

## $s v \in$ atoms

$(\bigwedge(\operatorname{map})(\lambda v$.
$\neg$ (Atom (Operator $t$ (index (operators-of $(\varphi$ (prob-with-noop abs-prob)))
$o p)$ )
$\vee \neg($ Atom $($ State $($ Suc $t)($ index vs $v))))$
asses) $) \Longrightarrow$
valid-state-var sv
using length-operators
by (induction asses) (auto simp: simp add: cnf-to-dimacs.valid-state-var.simps)

## end

lemma atoms-And-append: atoms $(\bigwedge(a s 1 @ a s 2))=$ atoms $(\bigwedge$ as1 $) \cup$ atoms (へas2)
by (induction as1) auto
context sat-solve-sasp

## begin

lemma encode-operator-effect-valid:
$s v \in$ atoms (encode-operator-effect ( $\varphi$ (prob-with-noop abs-prob)) top) $\Longrightarrow$
$t<h \Longrightarrow o p \in \operatorname{set}($ operators-of $(\varphi($ prob-with-noop abs-prob $))) \Longrightarrow$ valid-state-var sv
by (force simp: encode-operator-effect-def Let-def atoms-And-append intro!: encode-operator-effect-valid-1 encode-operator-effect-valid-2)

## end

lemma foldr-And: foldr $(\wedge)$ as $(\neg \perp)=(\bigwedge a s)$
by (induction as) auto
context sat-solve-sasp
begin
lemma encode-all-operator-effects-valid:
$t<$ Suc $h \Longrightarrow$
$s v \in$ atoms (encode-all-operator-effects ( $\varphi$ (prob-with-noop abs-prob)) (operators-of
$(\varphi($ prob-with-noop abs-prob $))) t) \Longrightarrow$
valid-state-var sv
unfolding encode-all-operator-effects-def foldr-And
by (force simp add: encode-operator-effect-valid)
lemma encode-operator-precondition-valid-1:
$t<h \Longrightarrow o p \in \operatorname{set}($ operators-of $(\varphi($ prob-with-noop abs-prob $))) \Longrightarrow$
$s v \in$ atoms
$(\bigwedge(\operatorname{map})(\lambda v$.
$\neg($ Atom $($ Operator $t$ (index (operators-of $(\varphi$ (prob-with-noop abs-prob)))
$o p))$ ) $\vee$ Atom $($ State $t(f v)))$
asses) $) \Longrightarrow$
valid-state-var sv
using length-operators
by (induction asses) (auto simp: simp add: cnf-to-dimacs.valid-state-var.simps)
lemma encode-operator-precondition-valid:
$s v \in$ atoms (encode-operator-precondition $(\varphi$ (prob-with-noop abs-prob)) top) $\Longrightarrow$
$t<h \Longrightarrow o p \in \operatorname{set}($ operators-of $(\varphi($ prob-with-noop abs-prob $))) \Longrightarrow$ valid-state-var sv
by (force simp: encode-operator-precondition-def Let-def

```
intro!: encode-operator-precondition-valid-1)
```

lemma encode-all-operator-preconditions-valid:
$t<$ Suc $h \Longrightarrow$
$s v \in$ atoms (encode-all-operator-preconditions ( $\varphi$ (prob-with-noop abs-prob))
(operators-of $(\varphi($ prob-with-noop abs-prob $))) t) \Longrightarrow$ valid-state-var sv
unfolding encode-all-operator-preconditions-def foldr-And
by (force simp add: encode-operator-precondition-valid)
lemma encode-operators-valid:
$s v \in$ atoms $($ encode-operators $(\varphi$ (prob-with-noop abs-prob)) $t) \Longrightarrow t<S u c h$
$\Longrightarrow$ valid-state-var sv
unfolding encode-operators-def Let-def
by (force simp add: encode-all-operator-preconditions-valid encode-all-operator-effects-valid)
lemma encode-negative-transition-frame-axiom':

```
    \(t<h \Longrightarrow\)
    set deleting-operators \(\subseteq\) set (operators-of \((\varphi(\) prob-with-noop abs-prob \())) \Longrightarrow\)
    \(s v \in\) atoms
        \((\neg(\) Atom \((\) State \(t v\)-idx \())\)
                            \(\vee\) (Atom (State (Suc t) v-idx)
            \(\vee \bigvee\) (map ( \(\lambda\) op. Atom (Operator \(t\) (index (operators-of ( \(\varphi\) (prob-with-noop
abs-prob))) op ())
            deleting-operators))) \(\Longrightarrow\)
        valid-state-var sv
by (induction deleting-operators) (auto simp: length-operators[symmetric] cnf-to-dimacs.valid-state-var.simps
```

lemma encode-negative-transition-frame-axiom-valid:
$s v \in$ atoms (encode-negative-transition-frame-axiom ( $\varphi$ (prob-with-noop abs-prob))
$t v) \Longrightarrow t<h \Longrightarrow$ valid-state-var sv
unfolding encode-negative-transition-frame-axiom-def Let-def
apply (intro encode-negative-transition-frame-axiom'[of t])
by auto
lemma encode-positive-transition-frame-axiom-valid:
$s v \in$ atoms (encode-positive-transition-frame-axiom ( $\varphi$ (prob-with-noop abs-prob))
$t v) \Longrightarrow t<h \Longrightarrow$
valid-state-var sv
unfolding encode-positive-transition-frame-axiom-def Let-def
apply (intro encode-negative-transition-frame-axiom'[of t])
by auto
lemma encode-all-frame-axioms-valid:
$s v \in$ atoms $($ encode-all-frame-axioms $(\varphi$ (prob-with-noop abs-prob)) $t) \Longrightarrow t<$
Suc $h \Longrightarrow$
valid-state-var sv
unfolding encode-all-frame-axioms-def Let-def atoms-And-append
by (force simp add: encode-negative-transition-frame-axiom-valid encode-positive-transition-frame-axiom-vali
lemma encode-goal-state-valid:
sv $\in$ atoms (encode-goal-state Prob $t) \Longrightarrow t<S u c h \Longrightarrow$ valid-state-var sv
by (auto simp add: encode-state-variable-def Let-def encode-goal-state-def split: sat-plan-variable.splits bool.splits)
lemma encode-problem-valid:
$s v \in$ atoms (encode-problem ( $\varphi$ (prob-with-noop abs-prob)) $h$ ) $\Longrightarrow$ valid-state-var sv
unfolding encode-problem-def
using encode-initial-state-valid encode-operators-valid encode-all-frame-axioms-valid encode-goal-state-valid
by fastforce
lemma encode-interfering-operator-pair-exclusion-valid:
$s v \in$ atoms (encode-interfering-operator-pair-exclusion ( $\varphi$ (prob-with-noop abs-prob))
$\left.t o p_{1} o p_{2}\right) \Longrightarrow t<S u c h \Longrightarrow$
op $p_{1} \in$ set (operators-of $\left(\varphi\right.$ (prob-with-noop abs-prob))) $\Longrightarrow o p_{2} \in$ set
(operators-of $(\varphi($ prob-with-noop abs-prob $))) \Longrightarrow$
valid-state-var sv
by (auto simp: encode-interfering-operator-pair-exclusion-def Let-def length-operators [symmetric] cnf-to-dimacs.valid-state-var.simps)
lemma encode-interfering-operator-exclusion-valid:
$s v \in$ atoms (encode-interfering-operator-exclusion ( $\varphi$ (prob-with-noop abs-prob))
$t) \Longrightarrow t<$ Suc $h \Longrightarrow$
valid-state-var sv
unfolding encode-interfering-operator-exclusion-def Let-def foldr-And
by (force simp add: encode-interfering-operator-pair-exclusion-valid)
lemma encode-problem-with-operator-interference-exclusion-valid:
$s v \in$ atoms (encode-problem-with-operator-interference-exclusion ( $\varphi$ (prob-with-noop abs-prob)) $h) \Longrightarrow$ valid-state-var sv
unfolding encode-problem-with-operator-interference-exclusion-def
using encode-initial-state-valid encode-operators-valid encode-all-frame-axioms-valid
encode-goal-state-valid
encode-interfering-operator-exclusion-valid
by fastforce
lemma planning-by-cnf-dimacs-complete:
valid-plan $\pi s \Longrightarrow$ length $\pi s \leq h \Longrightarrow$
$\exists M . M \models$ map-formula var-to-dimacs $\left(\Phi_{\forall}(\varphi(\right.$ prob-with-noop abs-prob $\left.)) h\right)$
using valid-plan-then-is-serial-sol-encoded sat-plan-to-dimacs[OF encode-problem-with-operator-interference-exclusion-valid]
by meson
lemma planning-by-cnf-dimacs-sound:

```
    \(\mathcal{A} \vDash\) map-formula var-to-dimacs \(\left(\Phi_{\forall}(\varphi(\right.\) prob-with-noop abs-prob \(\left.)) t\right) \Longrightarrow\)
    valid-plan
        (decode-abs-plan
            (rem-noops
            (map ( \(\lambda\) op. \(\varphi_{O}{ }^{-1}\) (prob-with-noop abs-prob) op)
                (concat \(\left(\Phi^{-1}(\varphi(\right.\) prob-with-noop abs-prob \())(\mathcal{A}\) o var-to-dimacs \(\left.\left.\left.\left.) t\right)\right)\right)\right)\)
    using changing-atoms-works \({ }^{\prime}\)
    by (fastforce intro!: is-serial-sol-then-valid-plan-encoded)
end
```


### 12.1 Going from Formualae to DIMACS-like CNF

We now represent the CNF formulae into a very low-level representation that is reminiscent to the DIMACS representation, where a CNF formula is a list of list of integers.
fun disj-to-dimacs::nat formula $\Rightarrow$ int list where

```
    disj-to-dimacs \(\left(\varphi_{1} \vee \varphi_{2}\right)=\) disj-to-dimacs \(\varphi_{1} @\) disj-to-dimacs \(\varphi_{2}\)
```

| disj-to-dimacs $\perp=[]$
| disj-to-dimacs $($ Not $\perp)=[-1::$ int $, 1::$ int $]$
|disj-to-dimacs (Atom v) $=[$ int $v]$
$\mid$ disj-to-dimacs $(\operatorname{Not}($ Atom $v))=[-($ int $v)]$
fun cnf-to-dimacs::nat formula $\Rightarrow$ int list list where
cnf-to-dimacs $\left(\varphi_{1} \wedge \varphi_{2}\right)=$ cnf-to-dimacs $\varphi_{1} @$ cnf-to-dimacs $\varphi_{2}$
$\mid$ cnf-to-dimacs $d=[$ disj-to-dimacs $d]$
definition dimacs-lit-to-var $l \equiv$ nat (abs l)
definition find-max $(x s:: n a t ~ l i s t) \equiv($ fold $\max x s 1)$
lemma find-max-works:

```
\(x \in\) set \(x s \Longrightarrow x \leq\) find-max \(x s(\) is \(? P \Longrightarrow\) ? \(Q\) )
proof -
    have \(x \in\) set \(x s \Longrightarrow(x:: n a t) \leq(\) fold max \(x s m)\) for \(m\)
        unfolding max-def
        apply (induction xs arbitrary: \(m\) rule: rev-induct)
        using nat-le-linear
        by (auto dest: le-trans simp add:)
    thus ? \(P \Longrightarrow\) ? \(Q\)
        by (auto simp add: find-max-def max-def)
qed
```

fun formula-vars where
formula-vars $(\perp)=[] \mid$
formula-vars $($ Atom $k)=[k] \mid$
formula-vars $($ Not $F)=$ formula-vars $F \mid$
formula-vars (And $F G$ ) = formula-vars $F$ @ formula-vars $G \mid$
formula-vars (Imp $F G$ ) $=$ formula-vars $F @$ formula-vars $G \mid$
formula-vars (Or $F G$ ) $=$ formula-vars $F$ @ formula-vars $G$
lemma atoms-formula-vars: atoms $f=\operatorname{set}($ formula-vars $f$ ) by (induction f) auto
lemma max-var: $v \in$ atoms $(f:: n a t$ formula) $\Longrightarrow v \leq$ find-max (formula-vars $f$ ) using find-max-works by (simp add: atoms-formula-vars)
definition dimacs-max-var $c s \equiv$ find-max $(\operatorname{map}(f i n d-m a x o(\operatorname{map}(n a t o a b s)))$ cs)
lemma fold-max-ge: $b \leq a \Longrightarrow(b::$ nat $) \leq$ fold $(\lambda x m$. if $m \leq x$ then $x$ else $m$ ) ys $a$
by (induction ys arbitrary: a b) auto
lemma find-max-append: find-max $(x s @ y s)=\max ($ find-max $x s)$ (find-max ys) apply (simp only: Max.set-eq-fold[symmetric] append-Cons[symmetric] set-append find-max-def)
by (metis List.finite-set Max.union Un-absorb Un-insert-left Un-insert-right list.distinct(1) list.simps(15) set-empty)
definition dimacs-model::int list $\Rightarrow$ int list list $\Rightarrow$ bool where
dimacs-model ls cs $\equiv(\forall c \in$ set cs. $(\exists l \in$ set ls. $l \in$ set $c)) \wedge$
distinct (map dimacs-lit-to-var ls)
fun model-to-dimacs-model where
model-to-dimacs-model $M(v \# v s)=($ if $M v$ then int $v$ else $-($ int $v)) \#($ model-to-dimacs-model M vs)
| model-to-dimacs-model - [] = []
lemma model-to-dimacs-model-append:
set (model-to-dimacs-model $M(v s @ v s '))=$ set (model-to-dimacs-model M vs) $\cup$ set (model-to-dimacs-model M vs')
by (induction vs) auto
lemma upt-append-sing: xs @ $[x]=[a . .<n$-vars $] \Longrightarrow a<n$-vars $\Longrightarrow(x s=[a . .<n$-vars
$-1] \wedge x=n$-vars $-1 \wedge n$-vars $>0)$
by (induction $n$-vars) auto
lemma upt-eqD: upt $a b=$ upt $a b^{\prime} \Longrightarrow\left(b=b^{\prime} \vee b^{\prime} \leq a \vee b \leq a\right)$
by (induction b) (auto dest!: upt-append-sing split: if-splits)
lemma pos-in-model: $M n \Longrightarrow 0<n \Longrightarrow n<n$-vars $\Longrightarrow$ int $n \in$ set (model-to-dimacs-model $M[1 . .<n$-vars $])$
by (induction $n$-vars) (auto simp add: less-Suc-eq model-to-dimacs-model-append )
lemma neg-in-model: $\neg M n \Longrightarrow 0<n \Longrightarrow n<n$-vars $\Longrightarrow-($ int $n) \in$ set

```
(model-to-dimacs-model M [1..<n-vars])
    by (induction n-vars) (auto simp add: less-Suc-eq model-to-dimacs-model-append)
lemma in-model: 0<n\Longrightarrown<n-vars \Longrightarrow int n \in set (model-to-dimacs-model
M [1..<n-vars]) \vee - (int n) \in set (model-to-dimacs-model M [1..<n-vars])
    using pos-in-model neg-in-model
    by metis
lemma model-to-dimacs-model-all-vars:
    ( }\forallv\in\mathrm{ atoms f. 0 < v^v<n-vars) # is-cnf f בM}\modelsf
        (\foralln<n-vars. 0 < n\longrightarrow(int n\in set (model-to-dimacs-model M [(1::nat)..<n-vars])
V
    -(int n) \in set (model-to-dimacs-model M [(1::nat)..<n-vars])))
    using in-model neg-in-model pos-in-model
    by (auto simp add:le-less model-to-dimacs-model-append split: if-splits)
lemma cnf-And: set (cnf-to-dimacs (f1 ^ f2)) = set (cnf-to-dimacs f1) \cup set
(cnf-to-dimacs f2)
    by auto
lemma one-always-in:
    1<n-vars \Longrightarrow1\in set (model-to-dimacs-model M ([1..<n-vars])) \vee - 1 \in set
(model-to-dimacs-model M ([1..<n-vars]))
    by (induction n-vars) (auto simp add: less-Suc-eq model-to-dimacs-model-append)
lemma [simp]:(disj-to-dimacs (f1 \vee f2)) = (disj-to-dimacs f1) @ (disj-to-dimacs
f2)
    by auto
lemma [simp]:(atoms (f1\vee f2)) = atoms f1 \cup atoms f2
    by auto
lemma isdisj-disjD:(is-disj (f1 \vee f2)) \Longrightarrow is-disj f1 ^ is-disj f2
    by (cases f1; auto)
lemma disj-to-dimacs-sound:
    1<n-vars \Longrightarrow(\forallv\inatoms f.0<v^v<n-vars) \Longrightarrowis-disj f \LongrightarrowM\modelsf
    \Longrightarrow\existsl\inset (model-to-dimacs-model M [(1::nat)..<n-vars]).l\in set (disj-to-dimacs
f)
    apply(induction f)
    using neg-in-model pos-in-model one-always-in
    by (fastforce elim!: is-lit-plus.elims dest!: isdisj-disjD)+
lemma is-cnf-disj:is-cnf (f1\vee f\mathscr{L})\Longrightarrow(\bigwedgef.f1\veef2 =f\Longrightarrow \s-disjf\LongrightarrowP)\Longrightarrow
P
    by auto
lemma cnf-to-dimacs-disj:is-disj f \Longrightarrow cnf-to-dimacs f}=[\mathrm{ disj-to-dimacs f}
    by (induction f) auto
```


## lemma model-to-dimacs-model-all-clauses:

$1<n$-vars $\Longrightarrow(\forall v \in$ atoms $f .0<v \wedge v<n$-vars $) \Longrightarrow i s$-cnf $f \Longrightarrow M \models f \Longrightarrow$ $c \in$ set (cnf-to-dimacs $f) \Longrightarrow \exists l \in$ set (model-to-dimacs-model $M$ [(1::nat).. $<n$-vars]).
$l \in \operatorname{set} c$
proof(induction $f$ arbitrary: )
case (Notf)
then show? case
using in-model neg-in-model
by (fastforce elim!: is-lit-plus.elims)+
next
case (Orf1 f2)
then show?case
using cnf-to-dimacs-disj disj-to-dimacs-sound
by (elim is-cnf-disj, simp)
qed (insert in-model neg-in-model pos-in-model, auto)
lemma upt-eq-Cons-conv:
$(x \# x s=[i . .<j])=(i<j \wedge i=x \wedge[i+1 . .<j]=x s)$
using upt-eq-Cons-conv
by metis
lemma model-to-dimacs-model-append':
( model-to-dimacs-model $\left.M\left(v s @ v s^{\prime}\right)\right)=($ model-to-dimacs-model $M v s) @($ model-to-dimacs-model M vs')
by (induction vs) auto
lemma model-to-dimacs-neg-nin:
$n$-vars $\leq x \Longrightarrow$ int $x \notin$ set (model-to-dimacs-model $M[a . .<n$-vars $]$ )
by (induction n-vars arbitrary: a) (auto simp: model-to-dimacs-model-append')
lemma model-to-dimacs-pos-nin:
$n$-vars $\leq x \Longrightarrow-$ int $x \notin$ set (model-to-dimacs-model $M$ [a..<n-vars $])$
by (induction n-vars arbitrary: a) (auto simp: model-to-dimacs-model-append')
lemma int-cases2 ':
$z \neq 0 \Longrightarrow(\bigwedge n .0 \neq($ int $n) \Longrightarrow z=$ int $n \Longrightarrow P) \Longrightarrow(\bigwedge n .0 \neq-($ int $n) \Longrightarrow$
$z=-($ int $n) \Longrightarrow P) \Longrightarrow P$
by (metis (full-types) int-cases2)
lemma model-to-dimacs-model-distinct:
$1<n$-vars $\Longrightarrow$ distinct (map dimacs-lit-to-var (model-to-dimacs-model $M[1 . .<n$-vars $]$ ))
by (induction $n$-vars)
(fastforce elim!: int-cases2 ${ }^{\prime}$
simp add: dimacs-lit-to-var-def model-to-dimacs-model-append' model-to-dimacs-neg-nin model-to-dimacs-pos-nin)+
lemma model-to-dimacs-model-sound:
$1<n$-vars $\Longrightarrow(\forall v \in$ atoms $f .0<v \wedge v<n$-vars $) \Longrightarrow i s$-cnf $f \Longrightarrow M \models f \Longrightarrow$
dimacs-model (model-to-dimacs-model $M[(1:: n a t) . .<n$-vars $])($ cnf-to-dimacs f)
unfolding dimacs-model-def
using model-to-dimacs-model-all-vars model-to-dimacs-model-all-clauses model-to-dimacs-model-distinct
by auto
lemma model-to-dimacs-model-sound-exists:
$1<n$-vars $\Longrightarrow(\forall v \in$ atoms $f .0<v \wedge v<n$-vars $) \Longrightarrow i s$-cnf $f \Longrightarrow M \models f \Longrightarrow$
$\exists M$-dimacs. dimacs-model $M$-dimacs (cnf-to-dimacs f)
using model-to-dimacs-model-sound
by metis
definition dimacs-to-atom $::$ int $\Rightarrow$ nat formula where
dimacs-to-atom $l \equiv$ if $(l<0)$ then Not (Atom (nat (abs l))) else Atom (nat (abs l))
definition dimacs-to-disj::int list $\Rightarrow$ nat formula where
dimacs-to-disj $f \equiv \bigvee($ map dimacs-to-atom $f)$
definition dimacs-to-cnf::int list list $\Rightarrow$ nat formula where
dimacs-to-cnf $f \equiv \bigwedge$ map dimacs-to-disj $f$
definition dimacs-model-to-abs dimacs-M $M \equiv$
fold $(\lambda l M$. if $(l>0)$ then $M(($ nat $($ abs $l)):=$ True $)$ else $M(($ nat $($ abs $l)):=$ False $))$
dimacs-M M
lemma dimacs-model-to-abs-atom:
$0<x \Longrightarrow$ int $x \in$ set dimacs- $M \Longrightarrow$ distinct (map dimacs-lit-to-var dimacs- $M$ )
$\Longrightarrow$ dimacs-model-to-abs dimacs-M M x
proof (induction dimacs-M arbitrary: $M$ rule: rev-induct)
case (snoc a dimacs-M)
thus ?case
by (auto simp add: dimacs-model-to-abs-def dimacs-lit-to-var-def image-def)
qed auto
lemma dimacs-model-to-abs-atom':
$0<x \Longrightarrow-($ int $x) \in$ set dimacs- $M \Longrightarrow$ distinct (map dimacs-lit-to-var di-macs- $M) \Longrightarrow \neg$ dimacs-model-to-abs dimacs-M $M x$
proof (induction dimacs- $M$ arbitrary: $M$ rule: rev-induct)
case (snoc a dimacs-M)
thus ?case
by (auto simp add: dimacs-model-to-abs-def dimacs-lit-to-var-def image-def) qed auto
lemma model-to-dimacs-model-complete-disj:
( $\forall v \in$ atoms $f .0<v \wedge v<n$-vars $) \Longrightarrow i s$-disj $f \Longrightarrow$ distinct (map dimacs-lit-to-var dimacs-M)
$\Longrightarrow$ dimacs-model dimacs-M (cnf-to-dimacs $f) \Longrightarrow$ dimacs-model-to-abs di-macs-M $(\lambda$-. False $) \models f$

```
    by (induction f)
```

    ( fastforce elim!: is-lit-plus.elims dest!: isdisj-disjD
        simp: cnf-to-dimacs-disj dimacs-model-def dimacs-model-to-abs-atom \({ }^{\prime}\)
                        dimacs-model-to-abs-atom)+
    lemma model-to-dimacs-model-complete:
( $\forall v \in$ atoms $f .0<v \wedge v<n$-vars $) \Longrightarrow$ is-cnff $\Longrightarrow$ distinct (map dimacs-lit-to-var dimacs-M)
$\Longrightarrow$ dimacs-model dimacs-M (cnf-to-dimacs $f) \Longrightarrow$ dimacs-model-to-abs di-
macs-M $(\lambda$-. False $) \models f$
proof $($ induction $f)$
case (Notf)
then show? case
by (auto elim!: is-lit-plus.elims simp add: dimacs-model-to-abs-atom' dimacs-model-def)
next
case (Or f1 f2)
then show? case
using cnf-to-dimacs-disj model-to-dimacs-model-complete-disj
by (elim is-cnf-disj, simp add: dimacs-model-def)
qed (insert dimacs-model-to-abs-atom, auto simp: dimacs-model-def)
lemma model-to-dimacs-model-complete-max-var:
$(\forall v \in$ atoms $f .0<v) \Longrightarrow$ is-cnf $f \Longrightarrow$
dimacs-model dimacs-M (cnf-to-dimacs $f) \Longrightarrow$
dimacs-model-to-abs dimacs- $M(\lambda$-. False $) \models f$
using le-imp-less-Suc[OF max-var]
by (auto intro!: model-to-dimacs-model-complete simp: dimacs-model-def)
lemma model-to-dimacs-model-sound-max-var:
$(\forall v \in$ atoms $f .0<v) \Longrightarrow i s-c n f f \Longrightarrow M \models f \Longrightarrow$
dimacs-model (model-to-dimacs-model $M[(1:: n a t) . .<($ find-max (formula-vars
f) +2$)]$ )
(cnf-to-dimacs f)
using le-imp-less-Suc[unfolded Suc-eq-plus1, OF max-var]
by (fastforce intro!: model-to-dimacs-model-sound)
context sat-solve-sasp
begin
lemma [simp]: var-to-dimacs sv>0
by (cases sv) auto
lemma var-to-dimacs-pos:
$v \in$ atoms (map-formula var-to-dimacs $f$ ) $\Longrightarrow 0<v$
by (induction f) auto
lemma map-is-disj: is-disj $f \Longrightarrow$ is-disj (map-formula $F f$ )
by (induction f) (auto elim: is-lit-plus.elims)

```
lemma map-is-cnf:is-cnff\Longrightarrow is-cnf (map-formula F f)
    by (induction f) (auto elim: is-lit-plus.elims simp: map-is-disj)
lemma planning-dimacs-complete:
    valid-plan }\pis\Longrightarrow\mathrm{ length }\pis\leqh
    let cnf-formula =(map-formula var-to-dimacs
                                    (\Phi}\mp@subsup{|}{}{\prime}(\varphi(\mathrm{ prob-with-noop abs-prob)) h))
    in
        \existsdimacs-M. dimacs-model dimacs-M (cnf-to-dimacs cnf-formula)
    unfolding Let-def
    by (fastforce simp: var-to-dimacs-pos
            dest!: planning-by-cnf-dimacs-complete
            intro: model-to-dimacs-model-sound-max-var map-is-cnf
        is-cnf-encode-problem-with-operator-interference-exclusion
        is-valid-problem-sas-plus-then-strips-transformation-too
        noops-valid abs-prob-valid)
lemma planning-dimacs-sound:
    let cnf-formula =
    (map-formula var-to-dimacs
                ( }\mp@subsup{\Phi}{\forall}{}(\varphi(prob-with-noop abs-prob)) h)
    in
    dimacs-model dimacs-M (cnf-to-dimacs cnf-formula) \Longrightarrow
    valid-plan
            (decode-abs-plan
                (rem-noops
                        (map (\lambdaop. \varphiO}\mp@subsup{O}{}{-1}\mathrm{ (prob-with-noop abs-prob) op)
                            (concat
                            ( }\mp@subsup{\Phi}{}{-1}(\varphi\mathrm{ (prob-with-noop abs-prob)) ((dimacs-model-to-abs dimacs-M
(\lambda-. False)) o var-to-dimacs) h)))))
    by(fastforce simp: var-to-dimacs-pos Let-def
            intro: planning-by-cnf-dimacs-sound model-to-dimacs-model-complete-max-var
            map-is-cnf is-cnf-encode-problem-with-operator-interference-exclusion
                    is-valid-problem-sas-plus-then-strips-transformation-too abs-prob-valid
                    noops-valid)
end
```


## 13 Code Generation

We now generate SML code equivalent to the functions that encode a problem as a CNF formula and that decode the model of the given encodings into a plan.

```
lemma [code]:
    dimacs-model ls cs \equiv(list-all (\lambdac. list-ex (\lambdal. ListMem l c) ls) cs) }
                distinct (map dimacs-lit-to-var ls)
    unfolding dimacs-model-def
```

```
by (auto simp: list.pred-set ListMem-iff list-ex-iff )
```

```
definition
SASP-to-DIMACS h prob \equiv
    cnf-to-dimacs
        (map-formula
            (cnf-to-dimacs.var-to-dimacs (Suc h) (Suc (length (ast-problem.ast\delta prob))))
                    ( }\mp@subsup{\Phi}{\forall}{}(\varphi\mathrm{ (prob-with-noop (ast-problem.abs-prob prob))) h))
lemma planning-dimacs-complete-code:
    |ast-problem.well-formed prob;
        \forall\pi\inset (ast-problem.ast\delta prob). is-standard-operator' \pi;
        ast-problem.valid-plan prob \pis;
        length }\pis\leqh\rrbracket
    let cnf-formula =(SASP-to-DIMACS h prob) in
        \exists dimacs-M. dimacs-model dimacs-M cnf-formula
    unfolding SASP-to-DIMACS-def Let-def
    apply(rule sat-solve-sasp.planning-dimacs-complete[unfolded Let-def])
    apply unfold-locales
    by auto
definition SASP-to-DIMACS' h prob \equivSASP-to-DIMACS h (rem-implicit-pres-ops
prob)
lemma planning-dimacs-complete-code':
    |ast-problem.well-formed prob;
        (\bigwedgeop.op \in set (ast-problem.ast\delta prob) \Longrightarrow consistent-pres-op op);
        (\bigwedgeop.op \in set (ast-problem.ast\delta prob) \Longrightarrow is-standard-operator op);
        ast-problem.valid-plan prob \pis;
        length }\pis\leqh\rrbracket
    let cnf-formula =(SASP-to-DIMACS' h prob) in
        \exists dimacs-M. dimacs-model dimacs-M cnf-formula
    unfolding Let-def SASP-to-DIMACS'-def
    by (auto simp add: rem-implicit-pres-ops-valid-plan[symmetric] wf-ast-problem-def
        simp del: rem-implicit-pres.simps
        intro!: rem-implicit-pres-is-standard-operator'
                        planning-dimacs-complete-code[unfolded Let-def]
            rem-implicit-pres-ops-well-formed
        dest!: rem-implicit-pres-ops-in\deltaD)
```

A function that does the checks required by the completeness theorem above, and returns appropriate error messages if any of the checks fail.

```
definition
    encode h prob \equiv
    if ast-problem.well-formed prob then
        if ( }\forall\mathrm{ op }\in\mathrm{ set (ast-problem.ast prob). consistent-pres-op op) then
            if ( }\forall\mathrm{ op }\in\mathrm{ set (ast-problem.ast }\delta\mathrm{ prob). is-standard-operator op) then
            Inl (SASP-to-DIMACS' h prob)
            else
```

```
        Inr (STR "Error: Conditional effects!")
        else
            Inr (STR "Error: Preconditions inconsistent")
        else
            Inr (STR '"Error: Problem malformed!')
lemma encode-sound:
    \llbracketast-problem.valid-plan prob \pis; length \pis \leqh;
            encode h prob = Inl cnf-formula\rrbracket \Longrightarrow
            ( }\exists\mathrm{ dimacs-M. dimacs-model dimacs-M cnf-formula)
    unfolding encode-def
    by (auto split: if-splits simp: list.pred-set
        intro: planning-dimacs-complete-code'[unfolded Let-def])
lemma encode-complete:
    encode h prob = Inr err }
        \neg ( a s t - p r o b l e m . w e l l - f o r m e d ~ p r o b ~ \wedge ~ ( ~ \forall o p ~ \in ~ s e t ~ ( a s t - p r o b l e m . a s t \delta ~ p r o b ) . c o n s i s - ~
tent-pres-op op) ^
            (\forallop 的 (ast-problem.ast\delta prob). is-standard-operator op))
    unfolding encode-def
    by (auto split: if-splits simp: list.pred-set
        intro: planning-dimacs-complete-code'[unfolded Let-def])
definition match-pre where
    match-pre }\equiv\lambda(x,v) s. s x = Some v
definition match-pres where
        match-pres pres s \equiv\forallpre\inset pres. match-pre pre s
lemma match-pres-distinct:
    distinct (map fst pres) \Longrightarrow match-pres pres s \longleftrightarrow Map.map-of pres }\mp@subsup{\subseteq}{m}{}
    unfolding match-pres-def match-pre-def
    using map-le-def map-of-SomeD
    apply (auto split: prod.splits)
    apply fastforce
    using domI map-of-is-SomeI
    by smt
fun tree-map-of where
    tree-map-of updatea T [] = T
| tree-map-of updatea T T((v,a)#m)= updatea v a (tree-map-of updatea T m)
context Map
begin
abbreviation tree-map-of \({ }^{\prime} \equiv\) tree-map-of update
lemma tree-map-of-invar: invar \(T \Longrightarrow\) invar (tree-map-of' \(T\) pres)
by (induction pres) (auto simp add: invar-update)
```

lemma tree-map-of-works: lookup (tree-map-of' empty pres) $x=$ map-of pres $x$
by (induction pres) (auto simp: map-empty map-update[OF tree-map-of-invar[OF invar-empty]])
lemma tree-map-of-dom: dom (lookup (tree-map-of' empty pres)) $=$ dom (map-of pres)
by (induction pres) (auto simp: map-empty map-update[OF tree-map-of-invar[OF invar-empty]] tree-map-of-works)
end
lemma distinct-if-sorted: sorted $x s \Longrightarrow$ distinct xs
by (induction xs rule: induct-list012) auto
context Map-by-Ordered
begin
lemma tree-map-of-distinct: distinct (map fst (inorder (tree-map-of ' empty pres))) apply (induction pres) apply (clarsimp simp: map-empty inorder-empty)
using distinct-if-sorted invar-def invar-empty invar-update tree-map-of-invar by blast
end
lemma set-tree-intorder: set-tree $t=$ set (inorder $t)$
by (induction t) auto
lemma map-of-eq:
map-of xs = Map.map-of xs
by (induction xs) (auto simp: map-of-simps split: option.split)
lemma lookup-someD: lookup $T x=$ Some $y \Longrightarrow \exists p . p \in \operatorname{set}($ inorder $T) \wedge p=$ ( $x, y$ )
by (induction $T$ ) (auto split: if-splits)
lemma map-of-lookup: sorted 1 (inorder $T) \Longrightarrow$ Map.map-of $($ inorder $T)=$ lookup $T$ apply(induction $T$ )
apply (auto split: prod.splits intro!: map-le-antisym
simp: lookup-map-of map-add-Some-iff map-of-None2 sorted-wrt-append)
using lookup-someD
by (force simp: map-of-eq map-add-def map-le-def
split: option.splits)+
lemma map-le-cong: $(\bigwedge x . m 1 x=m 2 x) \Longrightarrow m 1 \subseteq_{m} s \longleftrightarrow m 2 \subseteq_{m} s$
by presburger
lemma match-pres-submap:

```
match-pres (inorder (M.tree-map-of' empty pres))s su Map.map-of pres }\mp@subsup{\subseteq}{m}{}
using match-pres-distinct[OF M.tree-map-of-distinct]
by (smt M.invar-def M.invar-empty M.tree-map-of-invar M.tree-map-of-works
map-le-cong map-of-eq map-of-lookup)
lemma [code]:
    SAS-Plus-Representation.is-operator-applicable-in s op \longleftrightarrow
        match-pres (inorder (M.tree-map-of' empty (SAS-Plus-Representation.precondition-of
op))) s
    by (simp add: match-pres-submap SAS-Plus-Representation.is-operator-applicable-in-def)
definition decode-DIMACS-model dimacs-M h prob \equiv
    (ast-problem.decode-abs-plan prob
        (rem-noops
            (map (\lambdaop. \varphiO}\mp@subsup{O}{}{-1}(\mathrm{ prob-with-noop (ast-problem.abs-prob prob)) op)
                    (concat
                        ( }\mp@subsup{\Phi}{}{-1
                        ((dimacs-model-to-abs dimacs-M (\lambda-. False)) o
                                (cnf-to-dimacs.var-to-dimacs (Suc h)
                            (Suc (length (ast-problem.ast\delta prob)))))
                            h)))))
```


## lemma planning-dimacs-sound-code:

```
        |ast-problem.well-formed prob;
        \forall\inset (ast-problem.ast\delta prob). is-standard-operator'}\pi\rrbracket
        let
            cnf-formula =(SASP-to-DIMACS h prob);
            decoded-plan = decode-DIMACS-model dimacs-M h prob
        in
            (dimacs-model dimacs-M cnf-formula }\longrightarrow\mathrm{ ast-problem.valid-plan prob de-
coded-plan)
    unfolding SASP-to-DIMACS-def decode-DIMACS-model-def Let-def
    apply(rule impI sat-solve-sasp.planning-dimacs-sound[unfolded Let-def])+
    apply unfold-locales
    by auto
```


## definition

```
    decode-DIMACS-model' dimacs-M h prob \equiv
        decode-DIMACS-model dimacs-M h (rem-implicit-pres-ops prob)
lemma planning-dimacs-sound-code':
    |ast-problem.well-formed prob;
    (\bigwedgeop.op }\in\mathrm{ set (ast-problem.ast }\mathrm{ prob) }\Longrightarrow\mathrm{ consistent-pres-op op);
    \forall\inset (ast-problem.ast\delta prob). is-standard-operator \pi\rrbracket \Longrightarrow
        let
            cnf-formula =(SASP-to-DIMACS' h prob);
            decoded-plan = decode-DIMACS-model' dimacs-M h prob
        in
            (dimacs-model dimacs-M cnf-formula }\longrightarrow\mathrm{ ast-problem.valid-plan prob de-
```

```
coded-plan)
    unfolding SASP-to-DIMACS'-def decode-DIMACS-model'-def
    apply(subst rem-implicit-pres-ops-valid-plan[symmetric])
    by(fastforce simp only: rem-implicit-pres-ops-valid-plan wf-ast-problem-def
        intro!: rem-implicit-pres-is-standard-operator'
            rem-implicit-pres-ops-well-formed
            rev-iffD2[OF - rem-implicit-pres-ops-valid-plan]
            planning-dimacs-sound-code wf-ast-problem.intro
        dest!: rem-implicit-pres-ops-in\deltaD)+
```

Checking if the model satisfies the formula takes the longest time in the decoding function. We reimplement that part using red black trees, which makes it 10 times faster, on average!

```
fun list-to-rbt :: int list \(\Rightarrow\) int rbt where
    list-to-rbt [] = Leaf
\(\mid\) list-to-rbt \((x \# x s)=\) insert-rbt \(x(\) list-to-rbt \(x s)\)
```

lemma inv-list-to-rbt: invc (list-to-rbt xs) $\wedge$ invh (list-to-rbt xs)
by (induction xs) (auto simp: rbt-def RBT.inv-insert)
lemma Tree2-list-to-rbt: Tree2.bst (list-to-rbt xs)
by (induction xs) (auto simp: RBT.bst-insert)
lemma set-list-to-rbt: Tree2.set-tree (list-to-rbt xs) $=$ set $x s$
by (induction xs) (simp add: RBT.set-tree-insert Tree2-list-to-rbt)+
The following
lemma dimacs-model-code[code]:
dimacs-model ls cs $\longleftrightarrow$
(let tls $=$ list-to-rbt ls in
$(\forall c \in$ set cs. size $($ inter-rbt $($ tls $)($ list-to-rbt $c)) \neq 0) \wedge$
distinct (map dimacs-lit-to-var ls))
using RBT.set-tree-inter[OF Tree2-list-to-rbt Tree2-list-to-rbt]
apply (auto simp: dimacs-model-def Let-def set-list-to-rbt inter-rbt-def)
apply (metis IntI RBT.set-empty empty-iff)
by (metis Tree2.eq-set-tree-empty disjoint-iff-not-equal)
definition
decode M h prob $\equiv$
if ast-problem.well-formed prob then
if ( $\forall$ op $\in$ set (ast-problem.ast $\delta$ prob). consistent-pres-op op) then
if $(\forall$ op $\in$ set (ast-problem.ast $\delta$ prob). is-standard-operator op) then
if (dimacs-model M (SASP-to-DIMACS' $h$ prob)) then
Inl (decode-DIMACS-model' M h prob)
else Inr (STR "Error: Model does not solve the problem!')
else
Inr (STR "Error: Conditional effects!'")
else
Inr (STR 'Error: Preconditions inconsistent')

```
    else
    Inr (STR '"Error: Problem malformed!')
lemma decode-sound:
    decode M h prob = Inl plan }
    ast-problem.valid-plan prob plan
unfolding decode-def
apply (auto split: if-splits simp: list.pred-set)
using planning-dimacs-sound-code'
by auto
lemma decode-complete:
    decode M h prob = Inr err }
        \neg ~ ( a s t - p r o b l e m . w e l l - f o r m e d ~ p r o b ~ \wedge ~
        ( }\forall\mathrm{ op }\in\mathrm{ set (ast-problem.ast j prob). consistent-pres-op op) }
        (\forall\pi\inset (ast-problem.ast\delta prob). is-standard-operator \pi)^
        dimacs-model M (SASP-to-DIMACS' h prob))
    unfolding decode-def
    by (auto split: if-splits simp: list.pred-set)
lemma [code]:
    ListMem x' []= False
    ListMem x' (x#xs) = (x'=x\vee ListMem x' xs )
    by (simp add: ListMem-iff)+
lemmas [code] = SASP-to-DIMACS-def ast-problem.abs-prob-def
        ast-problem.abs-ast-variable-section-def ast-problem.abs-ast-operator-section-def
        ast-problem.abs-ast-initial-state-def ast-problem.abs-range-map-def
        ast-problem.abs-ast-goal-def cnf-to-dimacs.var-to-dimacs.simps
        ast-problem.ast \delta-def ast-problem.astDom-def ast-problem.abs-ast-operator-def
            ast-problem.astI-def ast-problem.astG-def ast-problem.lookup-action-def
        ast-problem.I-def execute-operator-sas-plus-def ast-problem.decode-abs-plan-def
definition nat-opt-of-integer :: integer }=>\mathrm{ nat option where
    nat-opt-of-integer i=(if (i\geq0) then Some (nat-of-integer i) else None)
definition max-var :: int list }=>\mathrm{ int where
    max-var xs \equiv fold (\lambda(x::int) (y::int). if abs x\geqabs y then (abs x) else y)xs
(0::int)
export-code encode nat-of-integer integer-of-nat nat-opt-of-integer Inl Inr String.explode String.implode max-var concat char-of-nat Int.nat integer-of-int length int-of-integer in SML module-name exported file-prefix SASP-to-DIMACS
export-code decode nat-of-integer integer-of-nat nat-opt-of-integer Inl Inr String.explode String.implode max-var concat char-of-nat Int.nat integer-of-int length int-of-integer in SML module-name exported file-prefix decode-DIMACS-model
end
```


## References

[1] M. Abdulaziz and F. Kurz. Formally verified sat-based ai planning, 2020.
[2] H. A. Kautz and B. Selman. Planning as satisfiability. In ECAI, pages 359-363, 1992.
[3] J. Rintanen, K. Heljanko, and I. Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. Artif. Intell., 170(12-13):1031-1080, 2006.
[4] M. Wenzel. The Isabelle/Isar Reference Manual, 2018. https://isabelle. in.tum.de/doc/isar-ref.pdf.


[^0]:    *Author names are alphabetically ordered.

[^1]:    ${ }^{1}$ For the full reference on records see [4, 11.6, pp.260-265]

[^2]:    ${ }^{2}$ Function effect_to_assignments converts the operator effect to a list of assignments.

[^3]:    ${ }^{3}$ Lemmas effect__strips_iii_a, effect__strips_iii_b, and effect__strips_iii_ c (not shown).

[^4]:    ${ }^{4}$ More precisely, the index of the last element is length $\tau-1$ if $\tau$ is not empty which is however always true since the trace contains at least the initial state.

[^5]:    ${ }^{5}$ We append a suffix identifying the respective formalism to the the parameter names passed to the parameter names in the locale. This is necessary to avoid ambiguous names in the sublocale declarations. For example, without addition of suffixes the type for initial-of is ambiguous and will therefore not be bound to either strips-problem.initial-of or sas-plus-problem.initial-of. Isabelle in fact considers it to be a a free variable in this case. We also qualify the parent locales in the sublocale declarations by adding strips: and sas_plus: before the respective parent locale identifiers.

[^6]:    ${ }^{8}$ This part of the soundness proof is only treated very briefly in [3, theorem 3.1, p.1044]

[^7]:    ${ }^{9}$ This lemma is used in the proof but not shown.

[^8]:    ${ }^{10}$ It is helpful to remember at this point, that the trace elements of a solution contain the states reached by plan prefix execution (lemma ??).

[^9]:    ${ }^{11} \mathrm{Cf}$. [3, Theorem 3.1, p. 1044] for the construction of $\mathcal{A}$.

[^10]:    ${ }^{12}$ These propositions are shown in lemmas encode_problem_forall_step_decoded_plan_is_serializable_ii and encode_problem_forall_step_decoded_plan_is_serializable_i which have been omitted for brevity.

