State Spaces: The Locale Way

Norbert Schirmer

January 18, 2026

Contents

1 Introduction

These theories introduce a new command called **statespace**. It's usage is similar to **record**s. However, the command does not introduce a new type but an abstract specification based on the locale infrastructure. This leads to extra flexibility in composing state space components, in particular multiple inheritance and renaming of components.

The state space infrastructure basically manages the following things:

- distinctness of field names
- projections / injections from / to an abstract *value* type
- syntax translations for lookup and update, hiding the projections and injections
- simplification procedure for lookups / updates, similar to records

Overview In Section 2 we define distinctness of the nodes in a binary tree and provide the basic prover tools to support efficient distinctness reasoning for field names managed by state spaces. The state is represented as a function from (abstract) names to (abstract) values as introduced in Section 3. The basic setup for state spaces is in Section 4. Some syntax for lookup and updates is added in Section 5. Finally Section 6 explains the usage of state spaces by examples.

2 Distinctness of Names in a Binary Tree

theory *DistinctTreeProver*
imports *Main*
begin

A state space manages a set of (abstract) names and assumes that the names are distinct. The names are stored as parameters of a locale and distinctness as an assumption. The most common request is to proof distinctness of two given names. We maintain the names in a balanced binary tree and formulate a predicate that all nodes in the tree have distinct names. This setup leads to logarithmic certificates.

2.1 The Binary Tree

datatype $'a\ tree = Node\ 'a\ tree\ 'a\ bool\ 'a\ tree \mid Tip$

The boolean flag in the node marks the content of the node as deleted, without having to build a new tree. We prefer the boolean flag to an option type, so that the ML-layer can still use the node content to facilitate binary search in the tree. The ML code keeps the nodes sorted using the term order. We do not have to push ordering to the HOL level.

2.2 Distinctness of Nodes

primrec $\mathit{set\text{-}of} :: 'a\ tree \Rightarrow 'a\ set$
where
  $\mathit{set\text{-}of}\ Tip = \{\}$
$\mid \mathit{set\text{-}of}\ (Node\ l\ x\ d\ r) = (\mathit{if}\ d\ \mathit{then}\ \{\}\ \mathit{else}\ \{x\}) \cup \mathit{set\text{-}of}\ l \cup \mathit{set\text{-}of}\ r$

primrec $\mathit{all\text{-}distinct} :: 'a\ tree \Rightarrow bool$
where
  $\mathit{all\text{-}distinct}\ Tip = True$
$\mid \mathit{all\text{-}distinct}\ (Node\ l\ x\ d\ r) =$
    $((d \vee (x \notin \mathit{set\text{-}of}\ l \wedge x \notin \mathit{set\text{-}of}\ r)) \wedge$
      $\mathit{set\text{-}of}\ l \cap \mathit{set\text{-}of}\ r = \{\} \wedge$
      $\mathit{all\text{-}distinct}\ l \wedge \mathit{all\text{-}distinct}\ r)$

Given a binary tree t for which *all-distinct* holds, given two different nodes contained in the tree, we want to write a ML function that generates a logarithmic certificate that the content of the nodes is distinct. We use the following lemmas to achieve this.

lemma *all-distinct-left*: $\textit{all-distinct}\ (\textit{Node}\ l\ x\ b\ r) \Longrightarrow \textit{all-distinct}\ l$
⟨*proof*⟩

lemma *all-distinct-right*: $\textit{all-distinct}\ (\textit{Node}\ l\ x\ b\ r) \Longrightarrow \textit{all-distinct}\ r$
⟨*proof*⟩

lemma *distinct-left*: $\textit{all-distinct}\ (\textit{Node}\ l\ x\ \textit{False}\ r) \Longrightarrow y \in \textit{set-of}\ l \Longrightarrow x \neq y$
⟨*proof*⟩

lemma *distinct-right*: $\textit{all-distinct}\ (\textit{Node}\ l\ x\ \textit{False}\ r) \Longrightarrow y \in \textit{set-of}\ r \Longrightarrow x \neq y$
⟨*proof*⟩

lemma *distinct-left-right*:
$\textit{all-distinct}\ (\textit{Node}\ l\ z\ b\ r) \Longrightarrow x \in \textit{set-of}\ l \Longrightarrow y \in \textit{set-of}\ r \Longrightarrow x \neq y$
⟨*proof*⟩

lemma *in-set-root*: $x \in \textit{set-of}\ (\textit{Node}\ l\ x\ \textit{False}\ r)$
⟨*proof*⟩

lemma *in-set-left*: $y \in \textit{set-of}\ l \Longrightarrow\ y \in \textit{set-of}\ (\textit{Node}\ l\ x\ \textit{False}\ r)$
⟨*proof*⟩

lemma *in-set-right*: $y \in \textit{set-of}\ r \Longrightarrow\ y \in \textit{set-of}\ (\textit{Node}\ l\ x\ \textit{False}\ r)$
⟨*proof*⟩

lemma *swap-neq*: $x \neq y \Longrightarrow y \neq x$
⟨*proof*⟩

lemma *neq-to-eq-False*: $x{\neq}y \Longrightarrow (x{=}y){\equiv}\textit{False}$
⟨*proof*⟩

2.3 Containment of Trees

When deriving a state space from other ones, we create a new name tree which contains all the names of the parent state spaces and assume the predicate *all-distinct*. We then prove that the new locale interprets all parent locales. Hence we have to show that the new distinctness assumption on all names implies the distinctness assumptions of the parent locales. This proof is implemented in ML. We do this efficiently by defining a kind of containment check of trees by “subtraction”. We subtract the parent tree from the new tree. If this succeeds we know that *all-distinct* of the new tree implies *all-distinct* of the parent tree. The resulting certificate is of the order $n * \textit{log}\ m$ where n is the size of the (smaller) parent tree and m the size of the (bigger) new tree.

primrec *delete* :: $'a \Rightarrow 'a\ tree \Rightarrow 'a\ tree\ option$
where
$delete\ x\ Tip = None$
$|\ delete\ x\ (Node\ l\ y\ d\ r) = (case\ delete\ x\ l\ of$
$Some\ l' \Rightarrow$
$(case\ delete\ x\ r\ of$
$Some\ r' \Rightarrow Some\ (Node\ l'\ y\ (d \lor (x{=}y))\ r')$
$|\ None \Rightarrow Some\ (Node\ l'\ y\ (d \lor (x{=}y))\ r))$
$|\ None \Rightarrow$
$(case\ delete\ x\ r\ of$
$Some\ r' \Rightarrow Some\ (Node\ l\ y\ (d \lor (x{=}y))\ r')$
$|\ None \Rightarrow if\ x{=}y \land \neg d\ then\ Some\ (Node\ l\ y\ True\ r)$
$else\ None))$

lemma *delete-Some-set-of*: $delete\ x\ t = Some\ t' \Longrightarrow set\text{-}of\ t' \subseteq set\text{-}of\ t$
⟨*proof*⟩

lemma *delete-Some-all-distinct*:
$delete\ x\ t = Some\ t' \Longrightarrow all\text{-}distinct\ t \Longrightarrow all\text{-}distinct\ t'$
⟨*proof*⟩

lemma *delete-None-set-of-conv*: $delete\ x\ t = None = (x \notin set\text{-}of\ t)$
⟨*proof*⟩

lemma *delete-Some-x-set-of*:
$delete\ x\ t = Some\ t' \Longrightarrow x \in set\text{-}of\ t \land x \notin set\text{-}of\ t'$
⟨*proof*⟩

primrec *subtract* :: $'a\ tree \Rightarrow 'a\ tree \Rightarrow 'a\ tree\ option$
where
$subtract\ Tip\ t = Some\ t$
$|\ subtract\ (Node\ l\ x\ b\ r)\ t =$
$(case\ delete\ x\ t\ of$
$Some\ t' \Rightarrow (case\ subtract\ l\ t'\ of$
$Some\ t'' \Rightarrow subtract\ r\ t''$
$|\ None \Rightarrow None)$
$|\ None \Rightarrow None)$

lemma *subtract-Some-set-of-res*:
$subtract\ t_1\ t_2 = Some\ t \Longrightarrow set\text{-}of\ t \subseteq set\text{-}of\ t_2$
⟨*proof*⟩

lemma *subtract-Some-set-of*:
$subtract\ t_1\ t_2 = Some\ t \Longrightarrow set\text{-}of\ t_1 \subseteq set\text{-}of\ t_2$
⟨*proof*⟩

lemma *subtract-Some-all-distinct-res*:
$\mathit{subtract}\ t_1\ t_2 = \mathit{Some}\ t \Longrightarrow \mathit{all\text{-}distinct}\ t_2 \Longrightarrow \mathit{all\text{-}distinct}\ t$
⟨*proof*⟩

lemma *subtract-Some-dist-res*:
$\mathit{subtract}\ t_1\ t_2 = \mathit{Some}\ t \Longrightarrow \mathit{set\text{-}of}\ t_1 \cap \mathit{set\text{-}of}\ t = \{\}$
⟨*proof*⟩

lemma *subtract-Some-all-distinct*:
$\mathit{subtract}\ t_1\ t_2 = \mathit{Some}\ t \Longrightarrow \mathit{all\text{-}distinct}\ t_2 \Longrightarrow \mathit{all\text{-}distinct}\ t_1$
⟨*proof*⟩

lemma *delete-left*:
assumes *dist*: $\mathit{all\text{-}distinct}\ (\mathit{Node}\ l\ y\ d\ r)$
assumes *del-l*: $\mathit{delete}\ x\ l = \mathit{Some}\ l'$
shows $\mathit{delete}\ x\ (\mathit{Node}\ l\ y\ d\ r) = \mathit{Some}\ (\mathit{Node}\ l'\ y\ d\ r)$
⟨*proof*⟩

lemma *delete-right*:
assumes *dist*: $\mathit{all\text{-}distinct}\ (\mathit{Node}\ l\ y\ d\ r)$
assumes *del-r*: $\mathit{delete}\ x\ r = \mathit{Some}\ r'$
shows $\mathit{delete}\ x\ (\mathit{Node}\ l\ y\ d\ r) = \mathit{Some}\ (\mathit{Node}\ l\ y\ d\ r')$
⟨*proof*⟩

lemma *delete-root*:
assumes *dist*: $\mathit{all\text{-}distinct}\ (\mathit{Node}\ l\ x\ \mathit{False}\ r)$
shows $\mathit{delete}\ x\ (\mathit{Node}\ l\ x\ \mathit{False}\ r) = \mathit{Some}\ (\mathit{Node}\ l\ x\ \mathit{True}\ r)$
⟨*proof*⟩

lemma *subtract-Node*:
assumes *del*: $\mathit{delete}\ x\ t = \mathit{Some}\ t'$
assumes *sub-l*: $\mathit{subtract}\ l\ t' = \mathit{Some}\ t''$
assumes *sub-r*: $\mathit{subtract}\ r\ t'' = \mathit{Some}\ t'''$
shows $\mathit{subtract}\ (\mathit{Node}\ l\ x\ \mathit{False}\ r)\ t = \mathit{Some}\ t'''$
⟨*proof*⟩

lemma *subtract-Tip*: $\mathit{subtract}\ \mathit{Tip}\ t = \mathit{Some}\ t$
⟨*proof*⟩

Now we have all the theorems in place that are needed for the certificate generating ML functions.

⟨*ML*⟩

end

3 State Space Representation as Function

theory *StateFun* **imports** *DistinctTreeProver*
begin

The state space is represented as a function from names to values. We neither fix the type of names nor the type of values. We define lookup and update functions and provide simprocs that simplify expressions containing these, similar to HOL-records.

The lookup and update function get constructor/destructor functions as parameters. These are used to embed various HOL-types into the abstract value type. Conceptually the abstract value type is a sum of all types that we attempt to store in the state space.

The update is actually generalized to a map function. The map supplies better compositionality, especially if you think of nested state spaces.

definition $K\text{-}statefun :: 'a \Rightarrow 'b \Rightarrow 'a$ **where** $K\text{-}statefun\ c\ x \equiv c$

lemma *K-statefun-apply* [*simp*]: $K\text{-}statefun\ c\ x = c$
⟨*proof*⟩

lemma *K-statefun-comp* [*simp*]: $(K\text{-}statefun\ c \circ f) = K\text{-}statefun\ c$
⟨*proof*⟩

lemma *K-statefun-cong* [*cong*]: $K\text{-}statefun\ c\ x = K\text{-}statefun\ c\ x$
⟨*proof*⟩

definition $lookup :: ('v \Rightarrow 'a) \Rightarrow 'n \Rightarrow ('n \Rightarrow 'v) \Rightarrow 'a$
where $lookup\ destr\ n\ s = destr\ (s\ n)$

definition $update ::$
$('v \Rightarrow 'a1) \Rightarrow ('a2 \Rightarrow 'v) \Rightarrow 'n \Rightarrow ('a1 \Rightarrow 'a2) \Rightarrow ('n \Rightarrow 'v) \Rightarrow ('n \Rightarrow 'v)$
where $update\ destr\ constr\ n\ f\ s = s(n := constr\ (f\ (destr\ (s\ n))))$

lemma *lookup-update-same*:
$(\bigwedge v.\ destr\ (constr\ v) = v) \Longrightarrow lookup\ destr\ n\ (update\ destr\ constr\ n\ f\ s) =$
$f\ (destr\ (s\ n))$
⟨*proof*⟩

lemma *lookup-update-id-same*:
$lookup\ destr\ n\ (update\ destr'\ id\ n\ (K\text{-}statefun\ (lookup\ id\ n\ s'))\ s) =$
$lookup\ destr\ n\ s'$
⟨*proof*⟩

lemma *lookup-update-other*:
$n \neq m \Longrightarrow lookup\ destr\ n\ (update\ destr'\ constr\ m\ f\ s) = lookup\ destr\ n\ s$
⟨*proof*⟩

lemma *id-id-cancel*: $id\ (id\ x) = x$
⟨*proof*⟩

lemma *destr-contstr-comp-id*: $(\bigwedge v.\ destr\ (constr\ v) = v) \Longrightarrow destr \circ constr = id$
⟨*proof*⟩

lemma *block-conj-cong*: $(P \wedge Q) = (P \wedge Q)$
⟨*proof*⟩

lemma *conj1-False*: $P \equiv False \Longrightarrow (P \wedge Q) \equiv False$
⟨*proof*⟩

lemma *conj2-False*: $Q \equiv False \Longrightarrow (P \wedge Q) \equiv False$
⟨*proof*⟩

lemma *conj-True*: $P \equiv True \Longrightarrow Q \equiv True \Longrightarrow (P \wedge Q) \equiv True$
⟨*proof*⟩

lemma *conj-cong*: $P \equiv P' \Longrightarrow Q \equiv Q' \Longrightarrow (P \wedge Q) \equiv (P' \wedge Q')$
⟨*proof*⟩

lemma *update-apply*: $(update\ destr\ constr\ n\ f\ s\ x) =$
$(if\ x{=}n\ then\ constr\ (f\ (destr\ (s\ n)))\ else\ s\ x)$
⟨*proof*⟩

lemma *ex-id*: $\exists x.\ id\ x = y$
⟨*proof*⟩

lemma *swap-ex-eq*:
$\exists s.\ f\ s = x \equiv True \Longrightarrow$
$\exists s.\ x = f\ s \equiv True$
⟨*proof*⟩

lemmas *meta-ext* = *eq-reflection* [*OF ext*]

lemma $update\ d\ c\ n\ (K\text{-}statespace\ (lookup\ d\ n\ s))\ s = s$
⟨*proof*⟩

end

4 Setup for State Space Locales

theory *StateSpaceLocale* **imports** *StateFun*
keywords *statespace* :: *thy-defn*
begin

⟨*ML*⟩

For every type that is to be stored in a state space, an instance of this locale is imported in order convert the abstract and concrete values.

locale *project-inject* =
 fixes *project* :: *'value* ⇒ *'a*
  and *inject* :: *'a* ⇒ *'value*
 assumes *project-inject-cancel* [*statefun-simp*]: *project* (*inject x*) = *x*
begin

lemma *ex-project* [*statefun-simp*]: ∃ *v*. *project v* = *x*
⟨*proof*⟩

lemma *project-inject-comp-id* [*statefun-simp*]: *project* ∘ *inject* = *id*
  ⟨*proof*⟩

lemma *project-inject-comp-cancel*[*statefun-simp*]: *f* ∘ *project* ∘ *inject* = *f*
  ⟨*proof*⟩

end

end

5 Syntax for State Space Lookup and Update

theory *StateSpaceSyntax*
imports *StateSpaceLocale*
begin

The state space syntax is kept in an extra theory so that you can choose if you want to use it or not.

syntax
  -statespace-lookup :: (*'a* ⇒ *'b*) ⇒ *'a* ⇒ *'c* (‹-·-› [*60*, *60*] *60*)
  -statespace-update :: (*'a* ⇒ *'b*) ⇒ *'a* ⇒ *'c* ⇒ (*'a* ⇒ *'b*)
  -statespace-updates :: (*'a* ⇒ *'b*) ⇒ *updbinds* ⇒ (*'a* ⇒ *'b*) (‹-<->› [*900*, *0*] *900*)

translations
  -statespace-updates f (*-updbinds b bs*) ==
    -statespace-updates (*-statespace-updates f b*) *bs*
  s<*x*:=*y*> == *-statespace-update s x y*

⟨*ML*⟩

end

6 Examples

theory *StateSpaceEx*
imports *StateSpaceLocale StateSpaceSyntax*
begin

Did you ever dream about records with multiple inheritance? Then you should definitely have a look at statespaces. They may be what you are dreaming of. Or at least almost ...

Isabelle allows to add new top-level commands to the system. Building on the locale infrastructure, we provide a command **statespace** like this:

statespace *vars* =
  $n::nat$
  $b::bool$

print-locale *vars-namespace*
print-locale *vars-valuetypes*
print-locale *vars*

This resembles a **record** definition, but introduces sophisticated locale infrastructure instead of HOL type schemes. The resulting context postulates two distinct names n and b and projection / injection functions that convert from abstract values to *nat* and *bool*. The logical content of the locale is:

locale $vars'$ =
  fixes $n::'name$ **and** $b::'name$
  assumes $distinct\ [n,\ b]$

  fixes $project\text{-}nat::'value \Rightarrow nat$ **and** $inject\text{-}nat::nat \Rightarrow 'value$
  assumes $\bigwedge n.\ project\text{-}nat\ (inject\text{-}nat\ n) = n$

  fixes $project\text{-}bool::'value \Rightarrow bool$ **and** $inject\text{-}bool::bool \Rightarrow 'value$
  assumes $\bigwedge b.\ project\text{-}bool\ (inject\text{-}bool\ b) = b$

The HOL predicate *distinct* describes distinctness of all names in the context. Locale $vars'$ defines the raw logical content that is defined in the state space locale. We also maintain non-logical context information to support the user:

- Syntax for state lookup and updates that automatically inserts the corresponding projection and injection functions.
- Setup for the proof tools that exploit the distinctness information and the cancellation of projections and injections in deductions and simplifications.

This extra-logical information is added to the locale in form of declarations, which associate the name of a variable to the corresponding projection and injection functions to handle the syntax transformations, and a link from the variable name to the corresponding distinctness theorem. As state spaces are merged or extended there are multiple distinctness theorems in the context. Our declarations take care that the link always points to the strongest distinctness assumption. With these declarations in place, a lookup can be written as $s{\cdot}n$, which is translated to *project-nat* $(s\ n)$, and an update as $s\langle n := \mathit{2}\rangle$, which is translated to $s(n := \textit{inject-nat } \mathit{2})$. We can now establish the following lemma:

lemma (**in** *vars*) *foo*: $s<n := \mathit{2}>\cdot b = s\cdot b$ $\langle \mathit{proof}\rangle$

Here the simplifier was able to refer to distinctness of b and n to solve the equation. The resulting lemma is also recorded in locale *vars* for later use and is automatically propagated to all its interpretations. Here is another example:

statespace $'a$ *varsX* = *NB*: *vars* $[n{=}N,\ b{=}B]$ + *vars* + $x{::}'a$

The state space *varsX* imports two copies of the state space *vars*, where one has the variables renamed to upper-case letters, and adds another variable x of type $'a$. This type is fixed inside the state space but may get instantiated later on, analogous to type parameters of an ML-functor. The distinctness assumption is now *distinct* $[N,\ B,\ n,\ b,\ x]$, from this we can derive both *distinct* $[N,\ B]$ and *distinct* $[n,\ b]$, the distinction assumptions for the two versions of locale *vars* above. Moreover we have all necessary projection and injection assumptions available. These assumptions together allow us to establish state space *varsX* as an interpretation of both instances of locale *vars*. Hence we inherit both variants of theorem *foo*: $s\langle N := \mathit{2}\rangle\cdot B = s\cdot B$ as well as $s\langle n := \mathit{2}\rangle\cdot b = s\cdot b$. These are immediate consequences of the locale interpretation action.

The declarations for syntax and the distinctness theorems also observe the morphisms generated by the locale package due to the renaming $n = N$:

lemma (**in** *varsX*) *foo*: $s\langle N := \mathit{2}\rangle\cdot x = s\cdot x$ $\langle \mathit{proof}\rangle$

To assure scalability towards many distinct names, the distinctness predicate is refined to operate on balanced trees. Thus we get logarithmic certificates for the distinctness of two names by the distinctness of the paths in the tree. Asked for the distinctness of two names, our tool produces the paths of the variables in the tree (this is implemented in Isabelle/ML, outside the logic) and returns a certificate corresponding to the different paths. Merging state spaces requires to prove that the combined distinctness assumption implies the distinctness assumptions of the components. Such a proof is of the order

$m \cdot \log n$, where n and m are the number of nodes in the larger and smaller tree, respectively.

We continue with more examples.

statespace *'a foo* =
  f::nat⇒nat
  a::int
  b::nat
  c::'a

lemma (**in** *foo*) *foo1*:
  shows $s\langle a := i\rangle \cdot a = i$
  ⟨*proof*⟩

lemma (**in** *foo*) *foo2*:
  shows $(s\langle a{:=}i\rangle)\cdot a = i$
  ⟨*proof*⟩

lemma (**in** *foo*) *foo3*:
  shows $(s\langle a{:=}i\rangle)\cdot b = s\cdot b$
  ⟨*proof*⟩

lemma (**in** *foo*) *foo4*:
  shows $(s\langle a{:=}i,b{:=}j,c{:=}k,a{:=}x\rangle) = (s\langle b{:=}j,c{:=}k,a{:=}x\rangle)$
  ⟨*proof*⟩

statespace *bar* =
  b::bool
  c::string

lemma (**in** *bar*) *bar1*:
  shows $(s\langle b{:=}True\rangle)\cdot c = s\cdot c$
  ⟨*proof*⟩

You can define a derived state space by inheriting existing state spaces, renaming of components if you like, and by declaring new components.

statespace (*'a,'b*) *loo* = *'a foo* + *bar* [*b=B,c=C*] +
  X::'b

lemma (**in** *loo*) *loo1*:
  shows $s\langle a{:=}i\rangle\cdot B = s\cdot B$
⟨*proof*⟩

statespace *'a dup* = *FA*: *'a foo* [*f=F, a=A*] + *'a foo* +
  x::int

lemma (**in** *dup*)
 shows $s{<}a := i{>}\cdot x = s\cdot x$
 ⟨*proof*⟩

lemma (**in** *dup*)
 shows $s{<}A := i{>}\cdot a = s\cdot a$
 ⟨*proof*⟩

lemma (**in** *dup*)
 shows $s{<}A := i{>}\cdot x = s\cdot x$
 ⟨*proof*⟩

There were known problems with syntax-declarations. They only worked when the context is already completely built. This is now overcome. e.g.:

locale *fooX* = *foo* +
 assumes $s{<}a{:=}i{>}\cdot b = k$

We can also put statespaces side-by-side by using ordinary **locale** expressions (instead of the **statespace**).

locale *side-by-side* = *foo* + *bar* **where** $b{=}B{::}'a$ **and** $c{=}C$ **for** B C

context *side-by-side*
begin

Simplification within one of the statespaces works as expected.

lemma $s{<}B := i{>}\cdot C = s\cdot C$
 ⟨*proof*⟩

lemma $s{<}a := i{>}\cdot b = s\cdot b$
 ⟨*proof*⟩

In contrast to the statespace *loo* there is no 'inter' statespace distinctness between the names of *foo* and *bar*.

end

Sharing of names in side-by-side statespaces is also possible as long as they are mapped to the same type.

statespace *vars1* = *n*::*nat* *m*::*nat*
statespace *vars2* = *n*::*nat* *k*::*nat*

locale *vars1-vars2* = *vars1* + *vars2*

context *vars1-vars2*
begin

Note that the distinctness theorem for *vars1* is selected here to do the proof.

lemma $s{<}n := i{>}\cdot m = s\cdot m$

$\langle proof \rangle$

Note that the distinctness theorem for *vars2* is selected here to do the proof.

lemma $s<n := i>\cdot k = s\cdot k$
$\langle proof \rangle$

Still there is no inter-statespace distinctness.

lemma $s<k := i>\cdot m = s\cdot m$

$\langle proof \rangle$
end

statespace *merge-vars1-vars2* = *vars1* + *vars2*

context *merge-vars1-vars2*
begin

When defining a statespace instead of a side-by-side locale we get the distinctness of all variables.

lemma $s<k := i>\cdot m = s\cdot m$
$\langle proof \rangle$
end

6.1 Benchmarks

Here are some bigger examples for benchmarking.

$\langle ML \rangle$

0.2s

statespace *benchmark100* = *A1*::*nat* *A2*::*nat* *A3*::*nat* *A4*::*nat* *A5*::*nat* *A6*::*nat* *A7*::*nat* *A8*::*nat* *A9*::*nat* *A10*::*nat* *A11*::*nat* *A12*::*nat* *A13*::*nat* *A14*::*nat* *A15*::*nat* *A16*::*nat* *A17*::*nat* *A18*::*nat* *A19*::*nat* *A20*::*nat* *A21*::*nat* *A22*::*nat* *A23*::*nat* *A24*::*nat* *A25*::*nat* *A26*::*nat* *A27*::*nat* *A28*::*nat* *A29*::*nat* *A30*::*nat* *A31*::*nat* *A32*::*nat* *A33*::*nat* *A34*::*nat* *A35*::*nat* *A36*::*nat* *A37*::*nat* *A38*::*nat* *A39*::*nat* *A40*::*nat* *A41*::*nat* *A42*::*nat* *A43*::*nat* *A44*::*nat* *A45*::*nat* *A46*::*nat* *A47*::*nat* *A48*::*nat* *A49*::*nat* *A50*::*nat* *A51*::*nat* *A52*::*nat* *A53*::*nat* *A54*::*nat* *A55*::*nat* *A56*::*nat* *A57*::*nat* *A58*::*nat* *A59*::*nat* *A60*::*nat* *A61*::*nat* *A62*::*nat* *A63*::*nat* *A64*::*nat* *A65*::*nat* *A66*::*nat* *A67*::*nat* *A68*::*nat* *A69*::*nat* *A70*::*nat* *A71*::*nat* *A72*::*nat* *A73*::*nat* *A74*::*nat* *A75*::*nat* *A76*::*nat* *A77*::*nat* *A78*::*nat* *A79*::*nat* *A80*::*nat* *A81*::*nat* *A82*::*nat* *A83*::*nat* *A84*::*nat* *A85*::*nat* *A86*::*nat* *A87*::*nat* *A88*::*nat* *A89*::*nat* *A90*::*nat* *A91*::*nat* *A92*::*nat* *A93*::*nat* *A94*::*nat* *A95*::*nat* *A96*::*nat* *A97*::*nat* *A98*::*nat* *A99*::*nat* *A100*::*nat*

2.4s

statespace *benchmark500* = *A1*::*nat* *A2*::*nat* *A3*::*nat* *A4*::*nat* *A5*::*nat* *A6*::*nat* *A7*::*nat* *A8*::*nat* *A9*::*nat* *A10*::*nat* *A11*::*nat* *A12*::*nat* *A13*::*nat*

$A14$::nat $A15$::nat $A16$::nat $A17$::nat $A18$::nat $A19$::nat $A20$::nat
$A21$::nat $A22$::nat $A23$::nat $A24$::nat $A25$::nat $A26$::nat $A27$::nat
$A28$::nat $A29$::nat $A30$::nat $A31$::nat $A32$::nat $A33$::nat $A34$::nat
$A35$::nat $A36$::nat $A37$::nat $A38$::nat $A39$::nat $A40$::nat $A41$::nat
$A42$::nat $A43$::nat $A44$::nat $A45$::nat $A46$::nat $A47$::nat $A48$::nat
$A49$::nat $A50$::nat $A51$::nat $A52$::nat $A53$::nat $A54$::nat $A55$::nat
$A56$::nat $A57$::nat $A58$::nat $A59$::nat $A60$::nat $A61$::nat $A62$::nat
$A63$::nat $A64$::nat $A65$::nat $A66$::nat $A67$::nat $A68$::nat $A69$::nat
$A70$::nat $A71$::nat $A72$::nat $A73$::nat $A74$::nat $A75$::nat $A76$::nat
$A77$::nat $A78$::nat $A79$::nat $A80$::nat $A81$::nat $A82$::nat $A83$::nat
$A84$::nat $A85$::nat $A86$::nat $A87$::nat $A88$::nat $A89$::nat $A90$::nat
$A91$::nat $A92$::nat $A93$::nat $A94$::nat $A95$::nat $A96$::nat $A97$::nat
$A98$::nat $A99$::nat $A100$::nat $A101$::nat $A102$::nat $A103$::nat $A104$::nat
$A105$::nat $A106$::nat $A107$::nat $A108$::nat $A109$::nat $A110$::nat $A111$::nat
$A112$::nat $A113$::nat $A114$::nat $A115$::nat $A116$::nat $A117$::nat $A118$::nat
$A119$::nat $A120$::nat $A121$::nat $A122$::nat $A123$::nat $A124$::nat $A125$::nat
$A126$::nat $A127$::nat $A128$::nat $A129$::nat $A130$::nat $A131$::nat $A132$::nat
$A133$::nat $A134$::nat $A135$::nat $A136$::nat $A137$::nat $A138$::nat $A139$::nat
$A140$::nat $A141$::nat $A142$::nat $A143$::nat $A144$::nat $A145$::nat $A146$::nat
$A147$::nat $A148$::nat $A149$::nat $A150$::nat $A151$::nat $A152$::nat $A153$::nat
$A154$::nat $A155$::nat $A156$::nat $A157$::nat $A158$::nat $A159$::nat $A160$::nat
$A161$::nat $A162$::nat $A163$::nat $A164$::nat $A165$::nat $A166$::nat $A167$::nat
$A168$::nat $A169$::nat $A170$::nat $A171$::nat $A172$::nat $A173$::nat $A174$::nat
$A175$::nat $A176$::nat $A177$::nat $A178$::nat $A179$::nat $A180$::nat $A181$::nat
$A182$::nat $A183$::nat $A184$::nat $A185$::nat $A186$::nat $A187$::nat $A188$::nat
$A189$::nat $A190$::nat $A191$::nat $A192$::nat $A193$::nat $A194$::nat $A195$::nat
$A196$::nat $A197$::nat $A198$::nat $A199$::nat $A200$::nat $A201$::nat $A202$::nat
$A203$::nat $A204$::nat $A205$::nat $A206$::nat $A207$::nat $A208$::nat $A209$::nat
$A210$::nat $A211$::nat $A212$::nat $A213$::nat $A214$::nat $A215$::nat $A216$::nat
$A217$::nat $A218$::nat $A219$::nat $A220$::nat $A221$::nat $A222$::nat $A223$::nat
$A224$::nat $A225$::nat $A226$::nat $A227$::nat $A228$::nat $A229$::nat $A230$::nat
$A231$::nat $A232$::nat $A233$::nat $A234$::nat $A235$::nat $A236$::nat $A237$::nat
$A238$::nat $A239$::nat $A240$::nat $A241$::nat $A242$::nat $A243$::nat $A244$::nat
$A245$::nat $A246$::nat $A247$::nat $A248$::nat $A249$::nat $A250$::nat $A251$::nat
$A252$::nat $A253$::nat $A254$::nat $A255$::nat $A256$::nat $A257$::nat $A258$::nat
$A259$::nat $A260$::nat $A261$::nat $A262$::nat $A263$::nat $A264$::nat $A265$::nat
$A266$::nat $A267$::nat $A268$::nat $A269$::nat $A270$::nat $A271$::nat $A272$::nat
$A273$::nat $A274$::nat $A275$::nat $A276$::nat $A277$::nat $A278$::nat $A279$::nat
$A280$::nat $A281$::nat $A282$::nat $A283$::nat $A284$::nat $A285$::nat $A286$::nat
$A287$::nat $A288$::nat $A289$::nat $A290$::nat $A291$::nat $A292$::nat $A293$::nat
$A294$::nat $A295$::nat $A296$::nat $A297$::nat $A298$::nat $A299$::nat $A300$::nat
$A301$::nat $A302$::nat $A303$::nat $A304$::nat $A305$::nat $A306$::nat $A307$::nat
$A308$::nat $A309$::nat $A310$::nat $A311$::nat $A312$::nat $A313$::nat $A314$::nat
$A315$::nat $A316$::nat $A317$::nat $A318$::nat $A319$::nat $A320$::nat $A321$::nat
$A322$::nat $A323$::nat $A324$::nat $A325$::nat $A326$::nat $A327$::nat $A328$::nat
$A329$::nat $A330$::nat $A331$::nat $A332$::nat $A333$::nat $A334$::nat $A335$::nat
$A336$::nat $A337$::nat $A338$::nat $A339$::nat $A340$::nat $A341$::nat $A342$::nat
$A343$::nat $A344$::nat $A345$::nat $A346$::nat $A347$::nat $A348$::nat $A349$::nat
$A350$::nat $A351$::nat $A352$::nat $A353$::nat $A354$::nat $A355$::nat $A356$::nat

A357::nat A358::nat A359::nat A360::nat A361::nat A362::nat A363::nat
A364::nat A365::nat A366::nat A367::nat A368::nat A369::nat A370::nat
A371::nat A372::nat A373::nat A374::nat A375::nat A376::nat A377::nat
A378::nat A379::nat A380::nat A381::nat A382::nat A383::nat A384::nat
A385::nat A386::nat A387::nat A388::nat A389::nat A390::nat A391::nat
A392::nat A393::nat A394::nat A395::nat A396::nat A397::nat A398::nat
A399::nat A400::nat A401::nat A402::nat A403::nat A404::nat A405::nat
A406::nat A407::nat A408::nat A409::nat A410::nat A411::nat A412::nat
A413::nat A414::nat A415::nat A416::nat A417::nat A418::nat A419::nat
A420::nat A421::nat A422::nat A423::nat A424::nat A425::nat A426::nat
A427::nat A428::nat A429::nat A430::nat A431::nat A432::nat A433::nat
A434::nat A435::nat A436::nat A437::nat A438::nat A439::nat A440::nat
A441::nat A442::nat A443::nat A444::nat A445::nat A446::nat A447::nat
A448::nat A449::nat A450::nat A451::nat A452::nat A453::nat A454::nat
A455::nat A456::nat A457::nat A458::nat A459::nat A460::nat A461::nat
A462::nat A463::nat A464::nat A465::nat A466::nat A467::nat A468::nat
A469::nat A470::nat A471::nat A472::nat A473::nat A474::nat A475::nat
A476::nat A477::nat A478::nat A479::nat A480::nat A481::nat A482::nat
A483::nat A484::nat A485::nat A486::nat A487::nat A488::nat A489::nat
A490::nat A491::nat A492::nat A493::nat A494::nat A495::nat A496::nat
A497::nat A498::nat A499::nat A500::nat

9.0s

statespace *benchmark1000* = A1::nat A2::nat A3::nat A4::nat A5::nat
A6::nat A7::nat A8::nat A9::nat A10::nat A11::nat A12::nat A13::nat
A14::nat A15::nat A16::nat A17::nat A18::nat A19::nat A20::nat
A21::nat A22::nat A23::nat A24::nat A25::nat A26::nat A27::nat
A28::nat A29::nat A30::nat A31::nat A32::nat A33::nat A34::nat
A35::nat A36::nat A37::nat A38::nat A39::nat A40::nat A41::nat
A42::nat A43::nat A44::nat A45::nat A46::nat A47::nat A48::nat
A49::nat A50::nat A51::nat A52::nat A53::nat A54::nat A55::nat
A56::nat A57::nat A58::nat A59::nat A60::nat A61::nat A62::nat
A63::nat A64::nat A65::nat A66::nat A67::nat A68::nat A69::nat
A70::nat A71::nat A72::nat A73::nat A74::nat A75::nat A76::nat
A77::nat A78::nat A79::nat A80::nat A81::nat A82::nat A83::nat
A84::nat A85::nat A86::nat A87::nat A88::nat A89::nat A90::nat
A91::nat A92::nat A93::nat A94::nat A95::nat A96::nat A97::nat
A98::nat A99::nat A100::nat A101::nat A102::nat A103::nat A104::nat
A105::nat A106::nat A107::nat A108::nat A109::nat A110::nat A111::nat
A112::nat A113::nat A114::nat A115::nat A116::nat A117::nat A118::nat
A119::nat A120::nat A121::nat A122::nat A123::nat A124::nat A125::nat
A126::nat A127::nat A128::nat A129::nat A130::nat A131::nat A132::nat
A133::nat A134::nat A135::nat A136::nat A137::nat A138::nat A139::nat
A140::nat A141::nat A142::nat A143::nat A144::nat A145::nat A146::nat
A147::nat A148::nat A149::nat A150::nat A151::nat A152::nat A153::nat
A154::nat A155::nat A156::nat A157::nat A158::nat A159::nat A160::nat
A161::nat A162::nat A163::nat A164::nat A165::nat A166::nat A167::nat
A168::nat A169::nat A170::nat A171::nat A172::nat A173::nat A174::nat
A175::nat A176::nat A177::nat A178::nat A179::nat A180::nat A181::nat

$A182$::nat $A183$::nat $A184$::nat $A185$::nat $A186$::nat $A187$::nat $A188$::nat
$A189$::nat $A190$::nat $A191$::nat $A192$::nat $A193$::nat $A194$::nat $A195$::nat
$A196$::nat $A197$::nat $A198$::nat $A199$::nat $A200$::nat $A201$::nat $A202$::nat
$A203$::nat $A204$::nat $A205$::nat $A206$::nat $A207$::nat $A208$::nat $A209$::nat
$A210$::nat $A211$::nat $A212$::nat $A213$::nat $A214$::nat $A215$::nat $A216$::nat
$A217$::nat $A218$::nat $A219$::nat $A220$::nat $A221$::nat $A222$::nat $A223$::nat
$A224$::nat $A225$::nat $A226$::nat $A227$::nat $A228$::nat $A229$::nat $A230$::nat
$A231$::nat $A232$::nat $A233$::nat $A234$::nat $A235$::nat $A236$::nat $A237$::nat
$A238$::nat $A239$::nat $A240$::nat $A241$::nat $A242$::nat $A243$::nat $A244$::nat
$A245$::nat $A246$::nat $A247$::nat $A248$::nat $A249$::nat $A250$::nat $A251$::nat
$A252$::nat $A253$::nat $A254$::nat $A255$::nat $A256$::nat $A257$::nat $A258$::nat
$A259$::nat $A260$::nat $A261$::nat $A262$::nat $A263$::nat $A264$::nat $A265$::nat
$A266$::nat $A267$::nat $A268$::nat $A269$::nat $A270$::nat $A271$::nat $A272$::nat
$A273$::nat $A274$::nat $A275$::nat $A276$::nat $A277$::nat $A278$::nat $A279$::nat
$A280$::nat $A281$::nat $A282$::nat $A283$::nat $A284$::nat $A285$::nat $A286$::nat
$A287$::nat $A288$::nat $A289$::nat $A290$::nat $A291$::nat $A292$::nat $A293$::nat
$A294$::nat $A295$::nat $A296$::nat $A297$::nat $A298$::nat $A299$::nat $A300$::nat
$A301$::nat $A302$::nat $A303$::nat $A304$::nat $A305$::nat $A306$::nat $A307$::nat
$A308$::nat $A309$::nat $A310$::nat $A311$::nat $A312$::nat $A313$::nat $A314$::nat
$A315$::nat $A316$::nat $A317$::nat $A318$::nat $A319$::nat $A320$::nat $A321$::nat
$A322$::nat $A323$::nat $A324$::nat $A325$::nat $A326$::nat $A327$::nat $A328$::nat
$A329$::nat $A330$::nat $A331$::nat $A332$::nat $A333$::nat $A334$::nat $A335$::nat
$A336$::nat $A337$::nat $A338$::nat $A339$::nat $A340$::nat $A341$::nat $A342$::nat
$A343$::nat $A344$::nat $A345$::nat $A346$::nat $A347$::nat $A348$::nat $A349$::nat
$A350$::nat $A351$::nat $A352$::nat $A353$::nat $A354$::nat $A355$::nat $A356$::nat
$A357$::nat $A358$::nat $A359$::nat $A360$::nat $A361$::nat $A362$::nat $A363$::nat
$A364$::nat $A365$::nat $A366$::nat $A367$::nat $A368$::nat $A369$::nat $A370$::nat
$A371$::nat $A372$::nat $A373$::nat $A374$::nat $A375$::nat $A376$::nat $A377$::nat
$A378$::nat $A379$::nat $A380$::nat $A381$::nat $A382$::nat $A383$::nat $A384$::nat
$A385$::nat $A386$::nat $A387$::nat $A388$::nat $A389$::nat $A390$::nat $A391$::nat
$A392$::nat $A393$::nat $A394$::nat $A395$::nat $A396$::nat $A397$::nat $A398$::nat
$A399$::nat $A400$::nat $A401$::nat $A402$::nat $A403$::nat $A404$::nat $A405$::nat
$A406$::nat $A407$::nat $A408$::nat $A409$::nat $A410$::nat $A411$::nat $A412$::nat
$A413$::nat $A414$::nat $A415$::nat $A416$::nat $A417$::nat $A418$::nat $A419$::nat
$A420$::nat $A421$::nat $A422$::nat $A423$::nat $A424$::nat $A425$::nat $A426$::nat
$A427$::nat $A428$::nat $A429$::nat $A430$::nat $A431$::nat $A432$::nat $A433$::nat
$A434$::nat $A435$::nat $A436$::nat $A437$::nat $A438$::nat $A439$::nat $A440$::nat
$A441$::nat $A442$::nat $A443$::nat $A444$::nat $A445$::nat $A446$::nat $A447$::nat
$A448$::nat $A449$::nat $A450$::nat $A451$::nat $A452$::nat $A453$::nat $A454$::nat
$A455$::nat $A456$::nat $A457$::nat $A458$::nat $A459$::nat $A460$::nat $A461$::nat
$A462$::nat $A463$::nat $A464$::nat $A465$::nat $A466$::nat $A467$::nat $A468$::nat
$A469$::nat $A470$::nat $A471$::nat $A472$::nat $A473$::nat $A474$::nat $A475$::nat
$A476$::nat $A477$::nat $A478$::nat $A479$::nat $A480$::nat $A481$::nat $A482$::nat
$A483$::nat $A484$::nat $A485$::nat $A486$::nat $A487$::nat $A488$::nat $A489$::nat
$A490$::nat $A491$::nat $A492$::nat $A493$::nat $A494$::nat $A495$::nat $A496$::nat
$A497$::nat $A498$::nat $A499$::nat $A500$::nat $A501$::nat $A502$::nat $A503$::nat
$A504$::nat $A505$::nat $A506$::nat $A507$::nat $A508$::nat $A509$::nat $A510$::nat
$A511$::nat $A512$::nat $A513$::nat $A514$::nat $A515$::nat $A516$::nat $A517$::nat
$A518$::nat $A519$::nat $A520$::nat $A521$::nat $A522$::nat $A523$::nat $A524$::nat

$A525$::nat $A526$::nat $A527$::nat $A528$::nat $A529$::nat $A530$::nat $A531$::nat
$A532$::nat $A533$::nat $A534$::nat $A535$::nat $A536$::nat $A537$::nat $A538$::nat
$A539$::nat $A540$::nat $A541$::nat $A542$::nat $A543$::nat $A544$::nat $A545$::nat
$A546$::nat $A547$::nat $A548$::nat $A549$::nat $A550$::nat $A551$::nat $A552$::nat
$A553$::nat $A554$::nat $A555$::nat $A556$::nat $A557$::nat $A558$::nat $A559$::nat
$A560$::nat $A561$::nat $A562$::nat $A563$::nat $A564$::nat $A565$::nat $A566$::nat
$A567$::nat $A568$::nat $A569$::nat $A570$::nat $A571$::nat $A572$::nat $A573$::nat
$A574$::nat $A575$::nat $A576$::nat $A577$::nat $A578$::nat $A579$::nat $A580$::nat
$A581$::nat $A582$::nat $A583$::nat $A584$::nat $A585$::nat $A586$::nat $A587$::nat
$A588$::nat $A589$::nat $A590$::nat $A591$::nat $A592$::nat $A593$::nat $A594$::nat
$A595$::nat $A596$::nat $A597$::nat $A598$::nat $A599$::nat $A600$::nat $A601$::nat
$A602$::nat $A603$::nat $A604$::nat $A605$::nat $A606$::nat $A607$::nat $A608$::nat
$A609$::nat $A610$::nat $A611$::nat $A612$::nat $A613$::nat $A614$::nat $A615$::nat
$A616$::nat $A617$::nat $A618$::nat $A619$::nat $A620$::nat $A621$::nat $A622$::nat
$A623$::nat $A624$::nat $A625$::nat $A626$::nat $A627$::nat $A628$::nat $A629$::nat
$A630$::nat $A631$::nat $A632$::nat $A633$::nat $A634$::nat $A635$::nat $A636$::nat
$A637$::nat $A638$::nat $A639$::nat $A640$::nat $A641$::nat $A642$::nat $A643$::nat
$A644$::nat $A645$::nat $A646$::nat $A647$::nat $A648$::nat $A649$::nat $A650$::nat
$A651$::nat $A652$::nat $A653$::nat $A654$::nat $A655$::nat $A656$::nat $A657$::nat
$A658$::nat $A659$::nat $A660$::nat $A661$::nat $A662$::nat $A663$::nat $A664$::nat
$A665$::nat $A666$::nat $A667$::nat $A668$::nat $A669$::nat $A670$::nat $A671$::nat
$A672$::nat $A673$::nat $A674$::nat $A675$::nat $A676$::nat $A677$::nat $A678$::nat
$A679$::nat $A680$::nat $A681$::nat $A682$::nat $A683$::nat $A684$::nat $A685$::nat
$A686$::nat $A687$::nat $A688$::nat $A689$::nat $A690$::nat $A691$::nat $A692$::nat
$A693$::nat $A694$::nat $A695$::nat $A696$::nat $A697$::nat $A698$::nat $A699$::nat
$A700$::nat $A701$::nat $A702$::nat $A703$::nat $A704$::nat $A705$::nat $A706$::nat
$A707$::nat $A708$::nat $A709$::nat $A710$::nat $A711$::nat $A712$::nat $A713$::nat
$A714$::nat $A715$::nat $A716$::nat $A717$::nat $A718$::nat $A719$::nat $A720$::nat
$A721$::nat $A722$::nat $A723$::nat $A724$::nat $A725$::nat $A726$::nat $A727$::nat
$A728$::nat $A729$::nat $A730$::nat $A731$::nat $A732$::nat $A733$::nat $A734$::nat
$A735$::nat $A736$::nat $A737$::nat $A738$::nat $A739$::nat $A740$::nat $A741$::nat
$A742$::nat $A743$::nat $A744$::nat $A745$::nat $A746$::nat $A747$::nat $A748$::nat
$A749$::nat $A750$::nat $A751$::nat $A752$::nat $A753$::nat $A754$::nat $A755$::nat
$A756$::nat $A757$::nat $A758$::nat $A759$::nat $A760$::nat $A761$::nat $A762$::nat
$A763$::nat $A764$::nat $A765$::nat $A766$::nat $A767$::nat $A768$::nat $A769$::nat
$A770$::nat $A771$::nat $A772$::nat $A773$::nat $A774$::nat $A775$::nat $A776$::nat
$A777$::nat $A778$::nat $A779$::nat $A780$::nat $A781$::nat $A782$::nat $A783$::nat
$A784$::nat $A785$::nat $A786$::nat $A787$::nat $A788$::nat $A789$::nat $A790$::nat
$A791$::nat $A792$::nat $A793$::nat $A794$::nat $A795$::nat $A796$::nat $A797$::nat
$A798$::nat $A799$::nat $A800$::nat $A801$::nat $A802$::nat $A803$::nat $A804$::nat
$A805$::nat $A806$::nat $A807$::nat $A808$::nat $A809$::nat $A810$::nat $A811$::nat
$A812$::nat $A813$::nat $A814$::nat $A815$::nat $A816$::nat $A817$::nat $A818$::nat
$A819$::nat $A820$::nat $A821$::nat $A822$::nat $A823$::nat $A824$::nat $A825$::nat
$A826$::nat $A827$::nat $A828$::nat $A829$::nat $A830$::nat $A831$::nat $A832$::nat
$A833$::nat $A834$::nat $A835$::nat $A836$::nat $A837$::nat $A838$::nat $A839$::nat
$A840$::nat $A841$::nat $A842$::nat $A843$::nat $A844$::nat $A845$::nat $A846$::nat
$A847$::nat $A848$::nat $A849$::nat $A850$::nat $A851$::nat $A852$::nat $A853$::nat
$A854$::nat $A855$::nat $A856$::nat $A857$::nat $A858$::nat $A859$::nat $A860$::nat
$A861$::nat $A862$::nat $A863$::nat $A864$::nat $A865$::nat $A866$::nat $A867$::nat

A868::*nat* *A869*::*nat* *A870*::*nat* *A871*::*nat* *A872*::*nat* *A873*::*nat* *A874*::*nat*
A875::*nat* *A876*::*nat* *A877*::*nat* *A878*::*nat* *A879*::*nat* *A880*::*nat* *A881*::*nat*
A882::*nat* *A883*::*nat* *A884*::*nat* *A885*::*nat* *A886*::*nat* *A887*::*nat* *A888*::*nat*
A889::*nat* *A890*::*nat* *A891*::*nat* *A892*::*nat* *A893*::*nat* *A894*::*nat* *A895*::*nat*
A896::*nat* *A897*::*nat* *A898*::*nat* *A899*::*nat* *A900*::*nat* *A901*::*nat* *A902*::*nat*
A903::*nat* *A904*::*nat* *A905*::*nat* *A906*::*nat* *A907*::*nat* *A908*::*nat* *A909*::*nat*
A910::*nat* *A911*::*nat* *A912*::*nat* *A913*::*nat* *A914*::*nat* *A915*::*nat* *A916*::*nat*
A917::*nat* *A918*::*nat* *A919*::*nat* *A920*::*nat* *A921*::*nat* *A922*::*nat* *A923*::*nat*
A924::*nat* *A925*::*nat* *A926*::*nat* *A927*::*nat* *A928*::*nat* *A929*::*nat* *A930*::*nat*
A931::*nat* *A932*::*nat* *A933*::*nat* *A934*::*nat* *A935*::*nat* *A936*::*nat* *A937*::*nat*
A938::*nat* *A939*::*nat* *A940*::*nat* *A941*::*nat* *A942*::*nat* *A943*::*nat* *A944*::*nat*
A945::*nat* *A946*::*nat* *A947*::*nat* *A948*::*nat* *A949*::*nat* *A950*::*nat* *A951*::*nat*
A952::*nat* *A953*::*nat* *A954*::*nat* *A955*::*nat* *A956*::*nat* *A957*::*nat* *A958*::*nat*
A959::*nat* *A960*::*nat* *A961*::*nat* *A962*::*nat* *A963*::*nat* *A964*::*nat* *A965*::*nat*
A966::*nat* *A967*::*nat* *A968*::*nat* *A969*::*nat* *A970*::*nat* *A971*::*nat* *A972*::*nat*
A973::*nat* *A974*::*nat* *A975*::*nat* *A976*::*nat* *A977*::*nat* *A978*::*nat* *A979*::*nat*
A980::*nat* *A981*::*nat* *A982*::*nat* *A983*::*nat* *A984*::*nat* *A985*::*nat* *A986*::*nat*
A987::*nat* *A988*::*nat* *A989*::*nat* *A990*::*nat* *A991*::*nat* *A992*::*nat* *A993*::*nat*
A994::*nat* *A995*::*nat* *A996*::*nat* *A997*::*nat* *A998*::*nat* *A999*::*nat* *A1000*::*nat*

lemma (**in** *benchmark100*) *test*: *s*<*A1* := *a*>·*A100* = *s*·*A100* ⟨*proof*⟩
lemma (**in** *benchmark500*) *test*: *s*<*A1* := *a*>·*A100* = *s*·*A100* ⟨*proof*⟩
lemma (**in** *benchmark1000*) *test*: *s*<*A1* := *a*>·*A100* = *s*·*A100* ⟨*proof*⟩

end