Sound Automation of Magic Wands

Thibault Dardinier[1], Gaurav Parthasarathy[1],
Noé Weeks[1], Peter Müller[1], and Alexander J. Summers[2]

[1] ETH zürich

[2] UBC

Sound Automation of Magic Wands

Binary connective in separation logic

Thibault Dardinier[1], Gaurav Parthasarathy[1],
Noé Weeks[1], Peter Müller[1], and Alexander J. Summers[2]

[1] ETH zürich

[2] UBC

Goal: Support the magic wand connective in automatic verifiers based on separation logic

Sound Automation of Magic Wands

Binary connective in separation logic

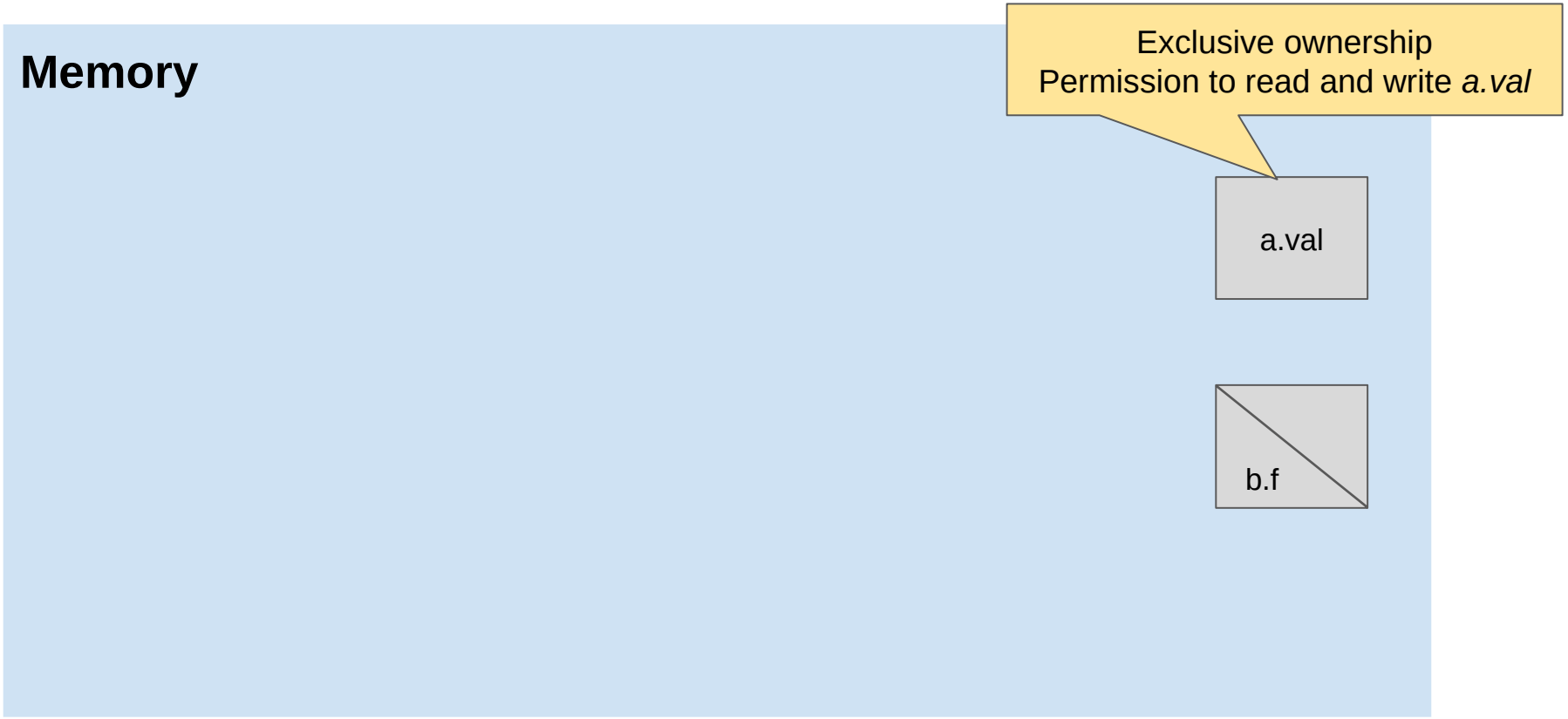
Thibault Dardinier[1], Gaurav Parthasarathy[1],
Noé Weeks[1], Peter Müller[1], and Alexander J. Summers[2]

[1] ETH zürich

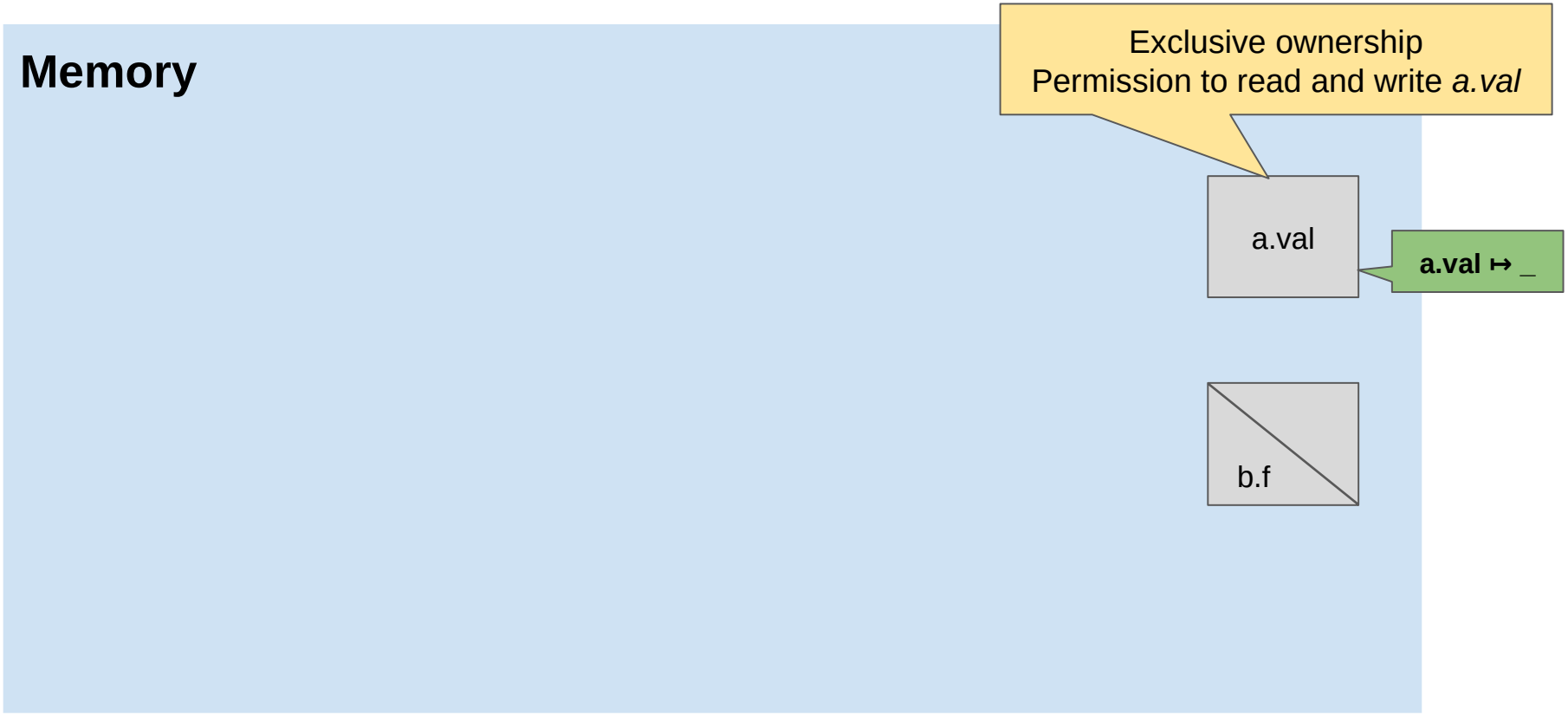
[2] UBC

Separation logic: Heap-manipulating programs and ownership

Separation logic: Heap-manipulating programs and ownership

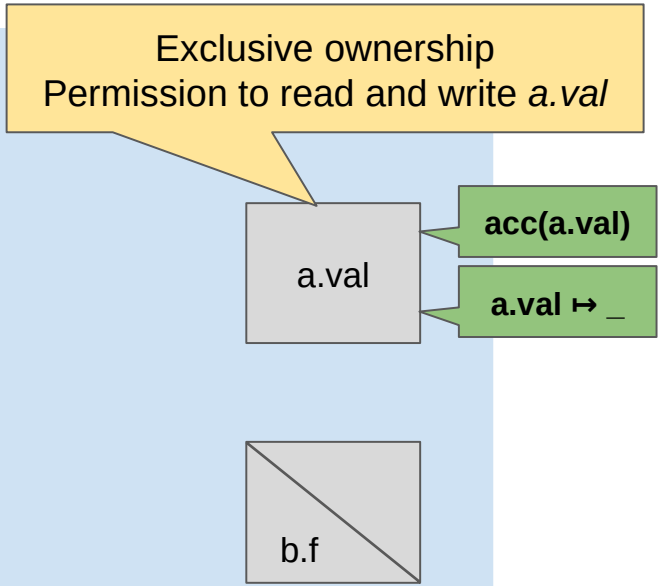


Separation logic: Heap-manipulating programs and ownership

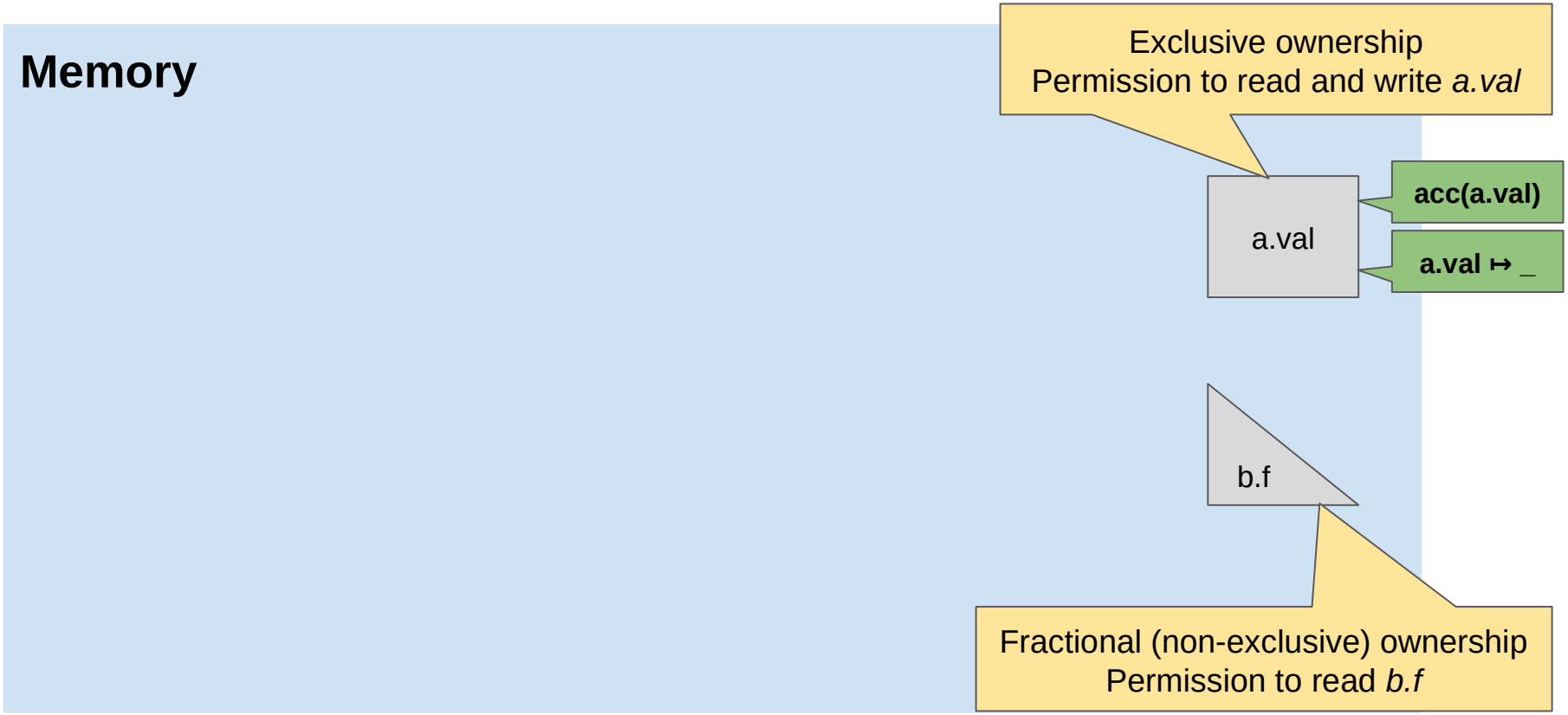


Separation logic: Heap-manipulating programs and ownership

Memory

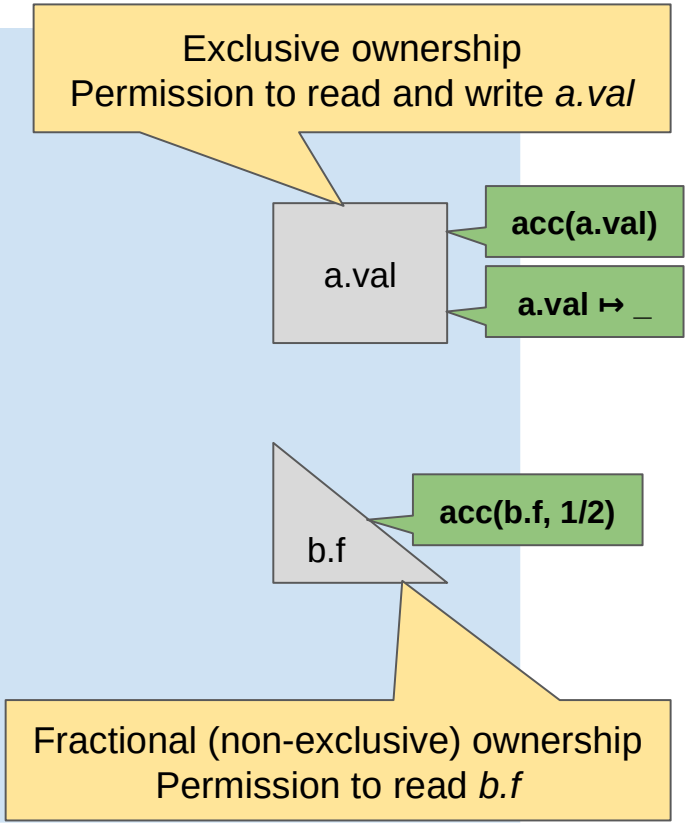


Separation logic: Heap-manipulating programs and ownership



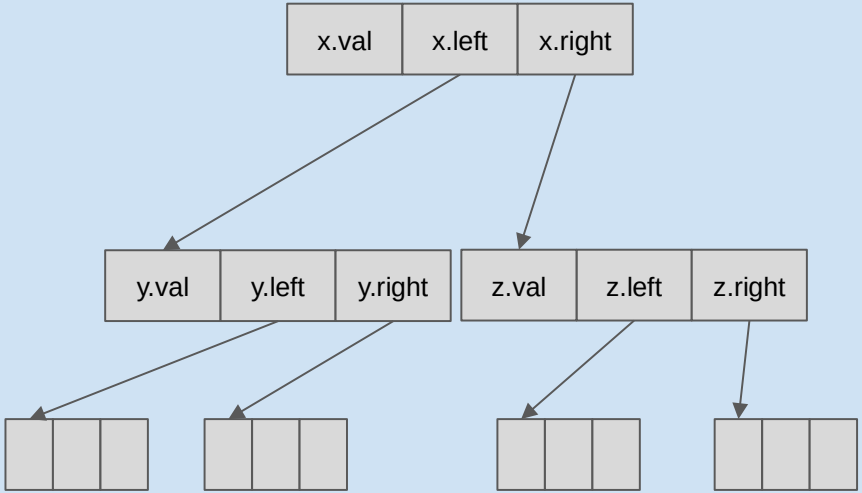
Separation logic: Heap-manipulating programs and ownership

Memory



Separation logic: Heap-manipulating programs and ownership

Memory



Exclusive ownership
Permission to read and write *a.val*

a.val

acc(a.val)

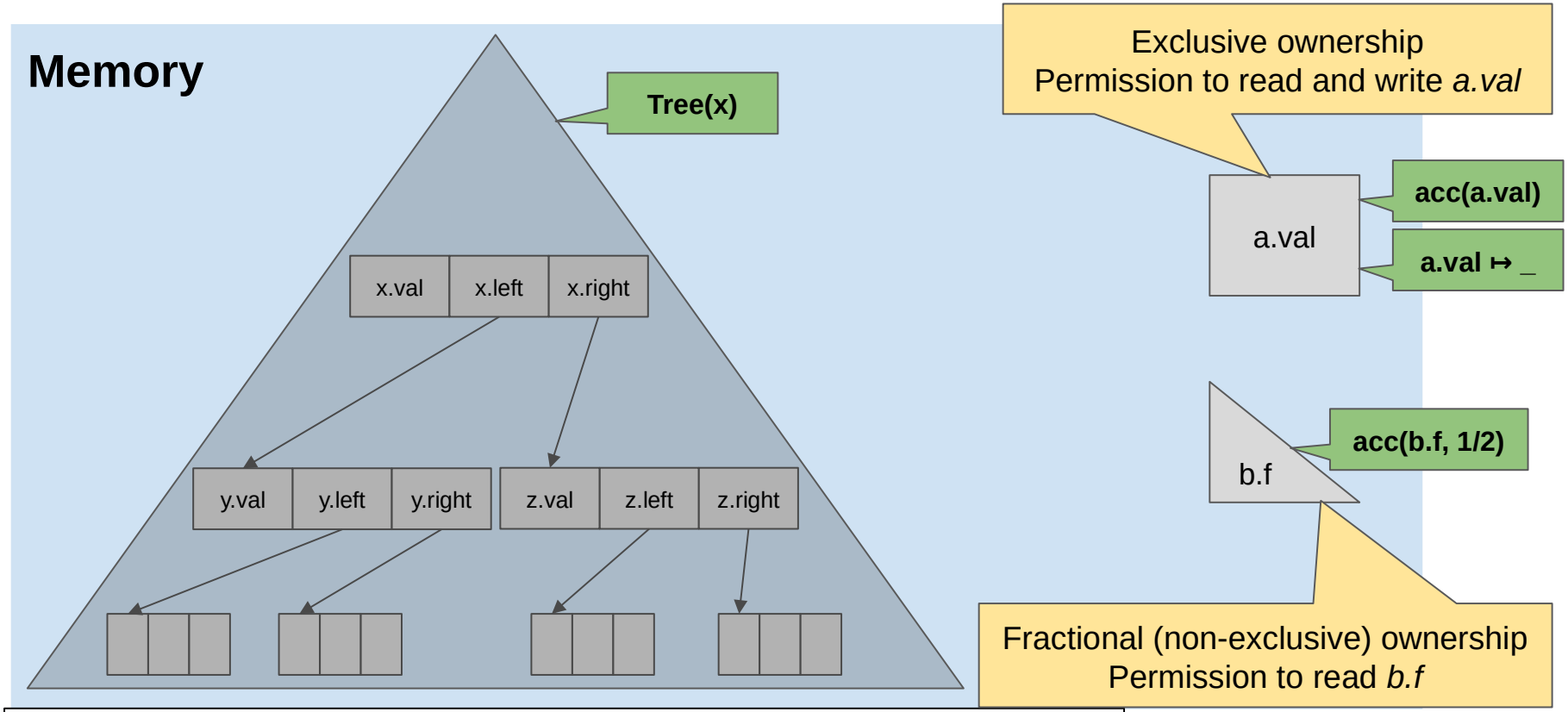
a.val ↦ _

b.f

acc(b.f, 1/2)

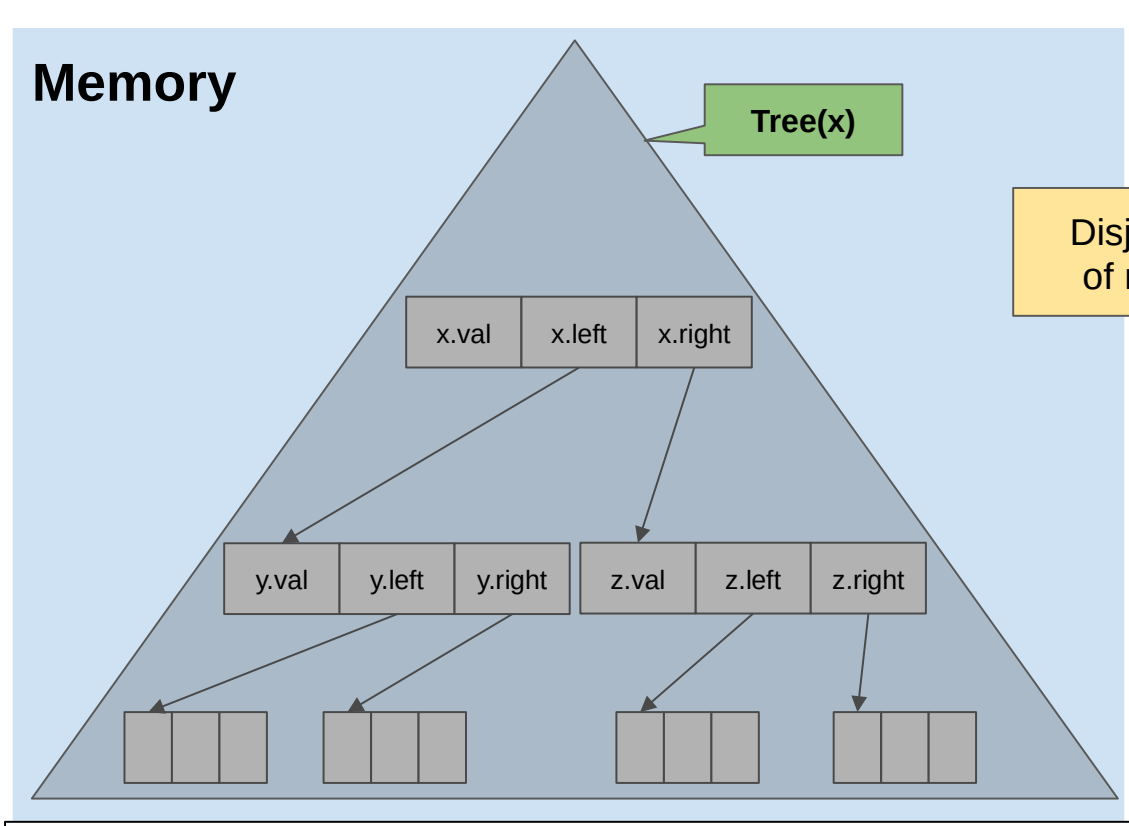
Fractional (non-exclusive) ownership
Permission to read *b.f*

Separation logic: Heap-manipulating programs and ownership




```
Tree(x: Ref) ≜
  acc(x.val) * acc(x.left) * acc(x.right) *
  (x.left != null ⇒ Tree(x.left)) * (x.right != null ⇒ Tree(x.right))
```

Separation logic: Heap-manipulating programs and ownership



Exclusive ownership
Permission to read and write *a.val*

Disjointness of memory

a.val

acc(a.val)

a.val ↦ _

*

b.f

acc(b.f, 1/2)

Fractional (non-exclusive) ownership
Permission to read *b.f*

```
Tree(x: Ref) ≜
  acc(x.val) * acc(x.left) * acc(x.right) *
  (x.left != null ⇒ Tree(x.left)) * (x.right != null ⇒ Tree(x.right))
```

Using magic wands

```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x


  while(y.left != null)
  invariant
  {
    y := y.left

  }

}
```

Using magic wands

Pointer to a binary tree

Computes its leftmost leaf

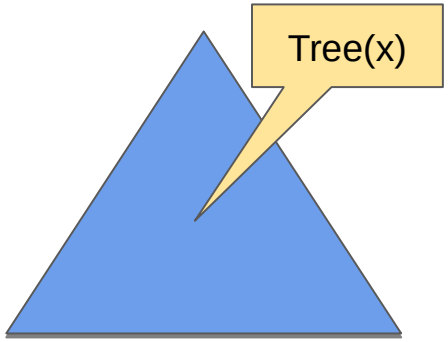
```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x


  while(y.left != null)
  invariant
  {
    y := y.left

  }

}
```

Using magic wands

Pointer to a binary tree

Computes its leftmost leaf

```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x


  while(y.left != null)
  invariant
  {
    y := y.left

  }

}
```

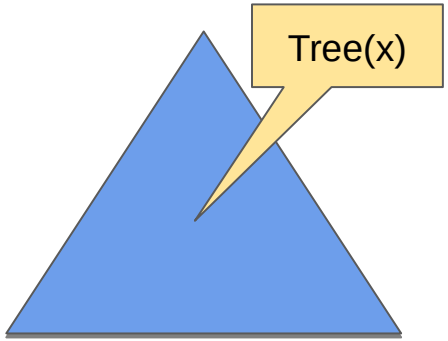


Using magic wands

Pointer to a binary tree

Computes its leftmost leaf

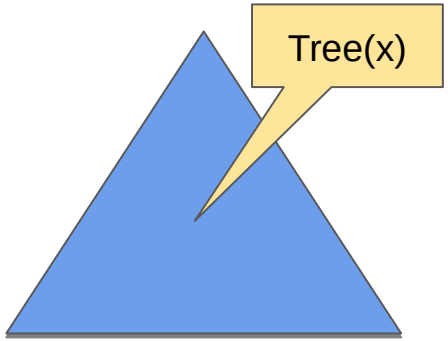
```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x


  while(y.left != null)
  invariant
  {
    y := y.left

  }

}
```



Using magic wands

```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x


  while(y.left != null)
  invariant
  {
    y := y.left

  }
}
```

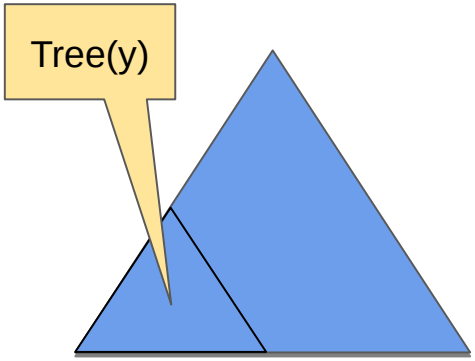
Need permission to justify read access



Using magic wands

```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x


  while(y.left != null)
  invariant Tree(y) *
  {
    y := y.left

  }
}
```

Need permission to justify read access

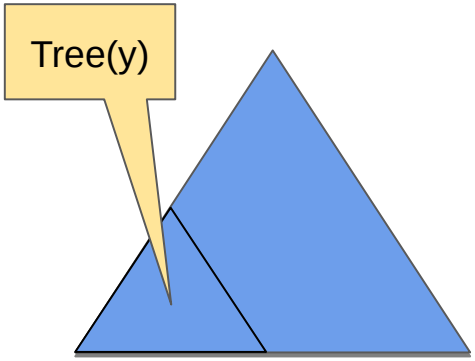


Using magic wands

```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x

  while(y.left != null)
  invariant Tree(y) *
  {
    y := y.left

  }

}
```

How to express the remaining ownership?

Need permission to justify read access

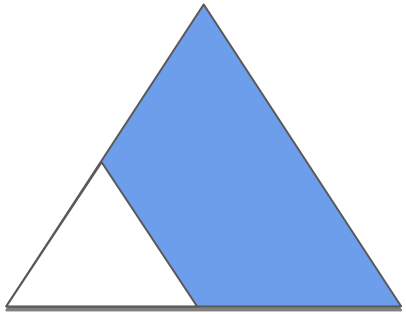


Using magic wands

```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x

  while(y.left != null)
  invariant Tree(y) *
  {
    y := y.left

  }

}
```

How to express the remaining ownership?

Need permission to justify read access

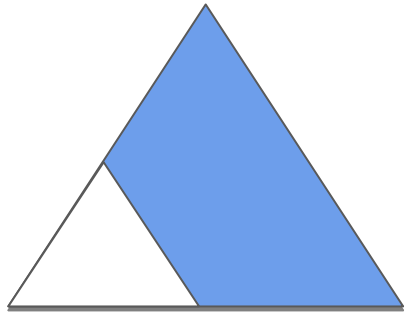


Using magic wands

```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x

  while(y.left != null)
  invariant Tree(y) *
  {
    y := y.left

  }

}
```

How to express the remaining ownership?

Need permission to justify read access



$$\mathrm{Tree}(y) \mathrel{-\!\!*} \mathrm{Tree}(x)$$

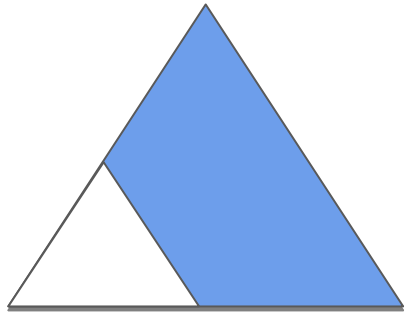
Magic wand

Using magic wands

```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x

  while(y.left != null)
  invariant Tree(y) * (Tree(y) -* Tree(x))
  {
    y := y.left

  }

}
```

How to express the remaining ownership?

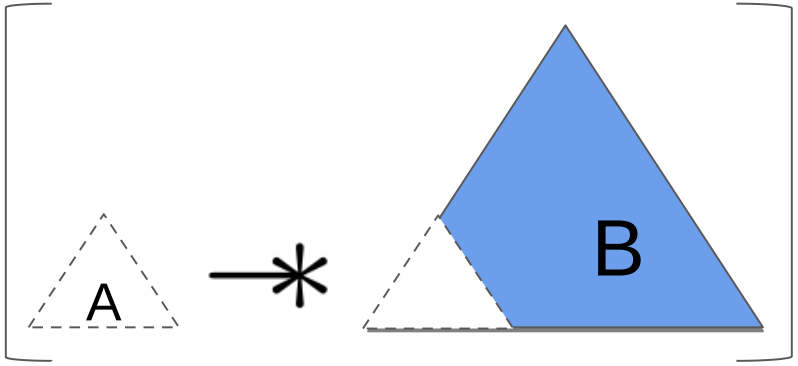
Need permission to justify read access



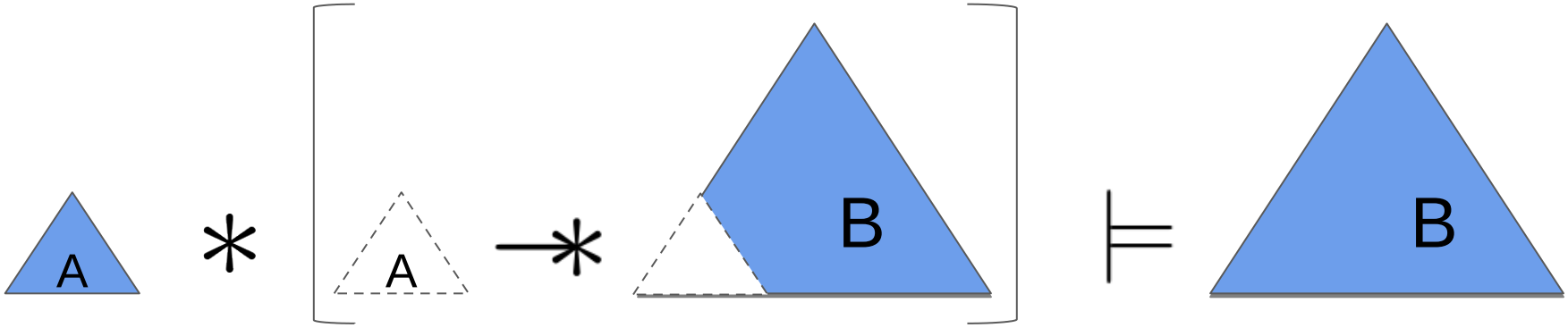
$$Tree(y) \mathrel{-\!\!*} Tree(x)$$

Magic wand

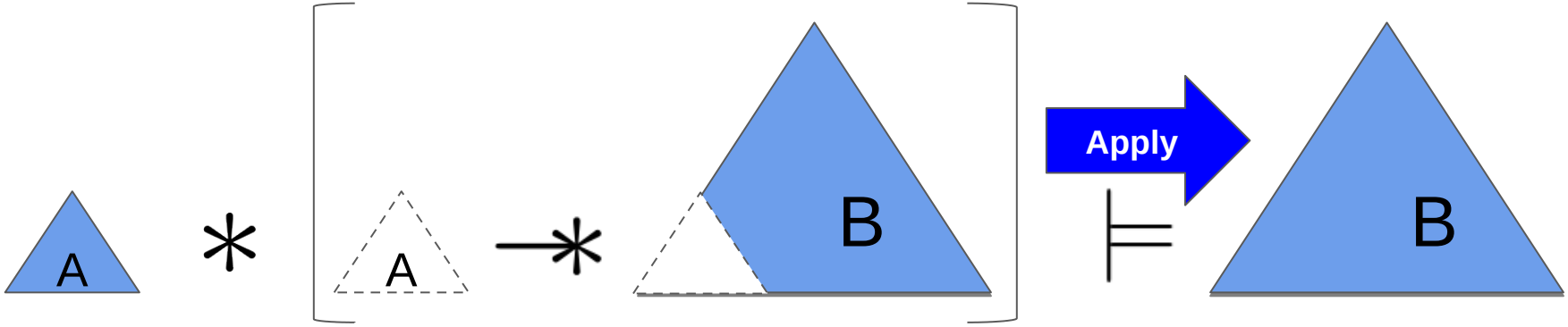
Background: Ghost operations to manipulate magic wands



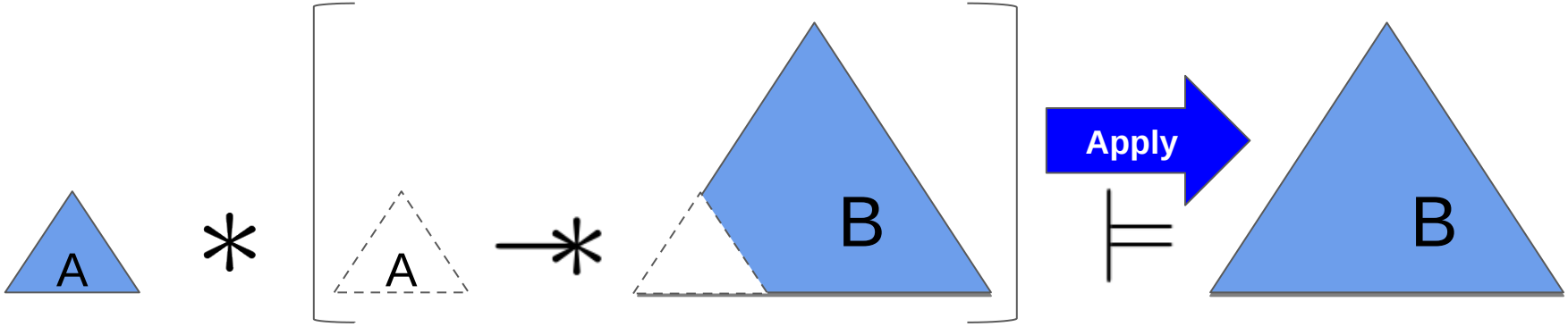
Background: Ghost operations to manipulate magic wands

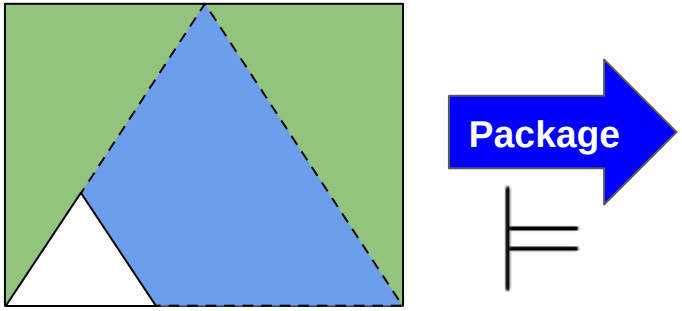


Background: Ghost operations to manipulate magic wands

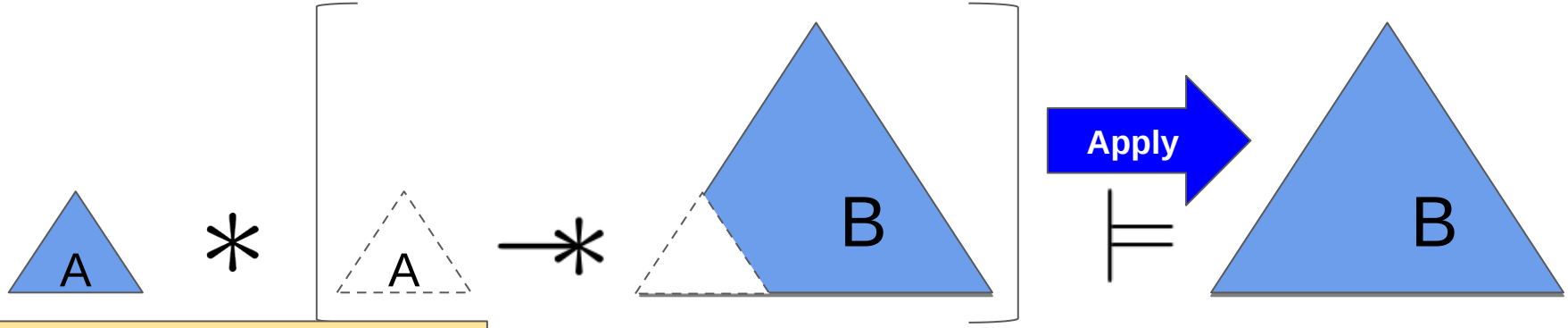


Background: Ghost operations to manipulate magic wands

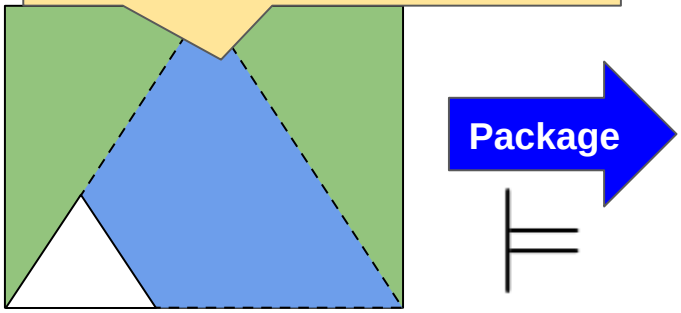




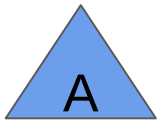
Background: Ghost operations to manipulate magic wands

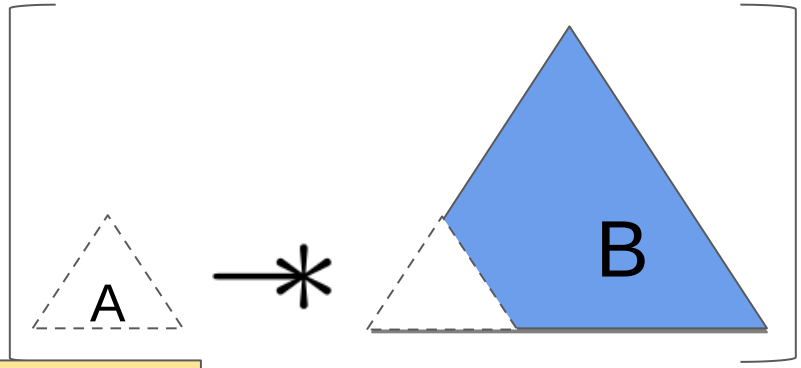


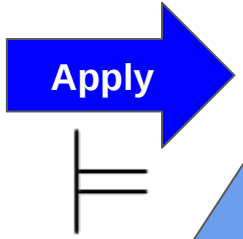
Footprint: Set of resources that, when combined with **A**, guarantee **B**



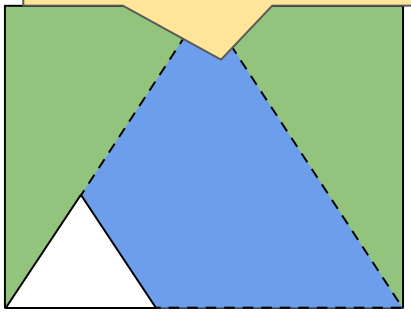
Background: Ghost operations to manipulate magic wands

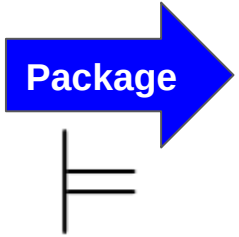


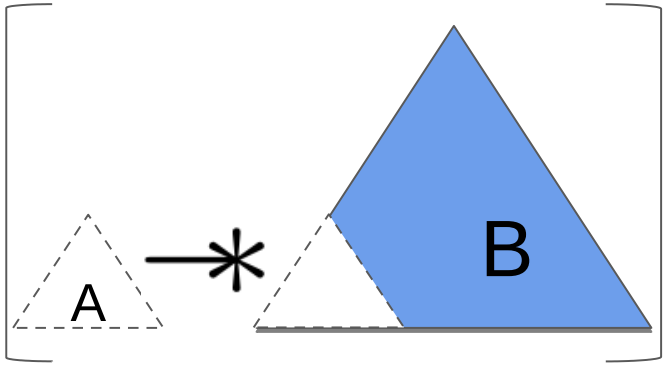


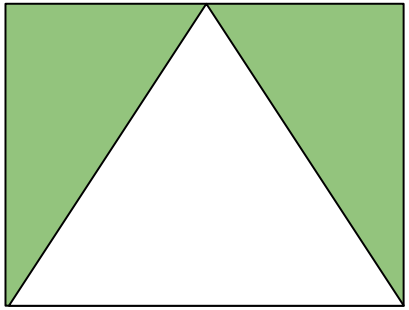


Footprint: Set of resources that, when combined with **A**, guarantee **B**



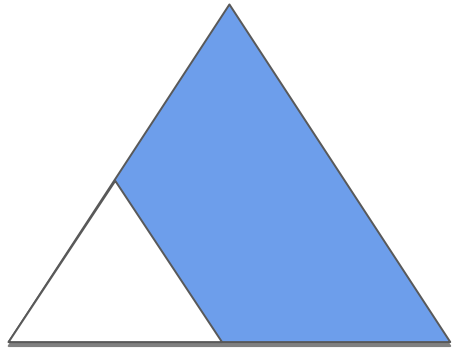






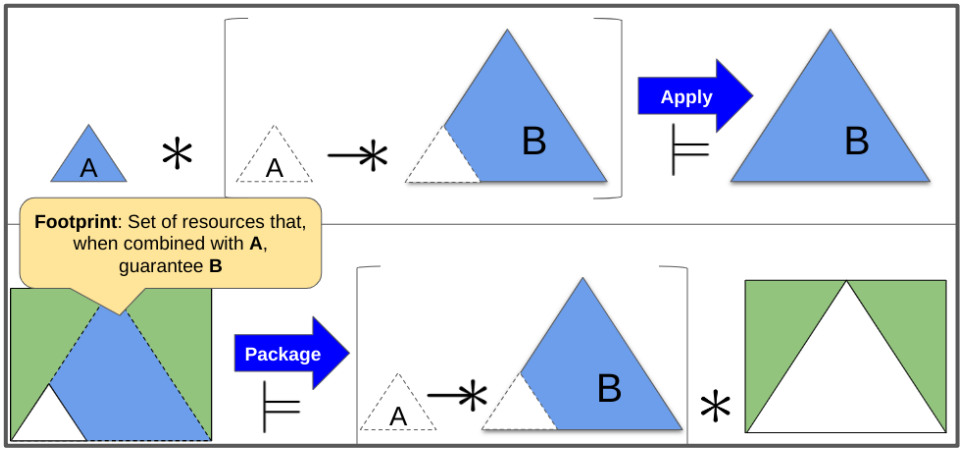
Using magic wands

```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x


  while(y.left != null)
  invariant Tree(y) ∗ (Tree(y) −∗ Tree(x))
  {
    y := y.left

  }

}
```

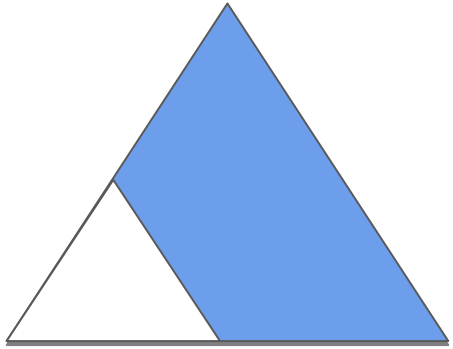


$$\mathrm{Tree}(y) \mathrel{-\!\!*} \mathrm{Tree}(x)$$

Using magic wands

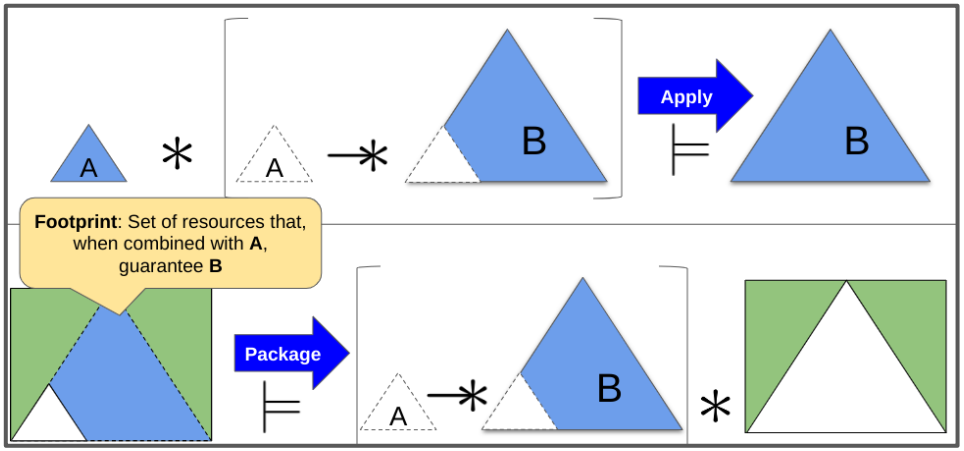



```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x


  while(y.left != null)
  invariant Tree(y) * (Tree(y) -* Tree(x))
  {
    y := y.left

  }

}
```

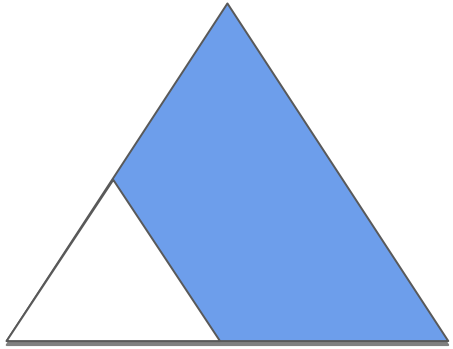


Tree(y) −∗ Tree(x)

Using magic wands

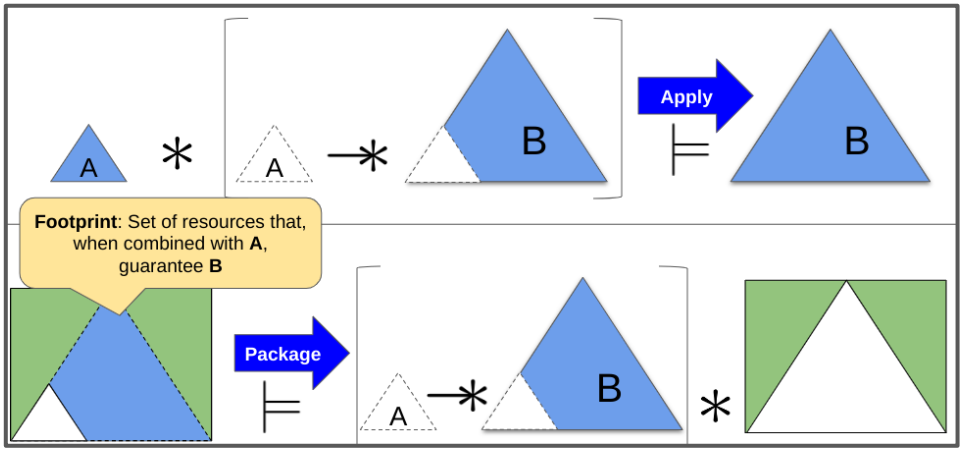



```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x


  while(y.left != null)
  invariant Tree(y) * (Tree(y) −∗ Tree(x))
  {
    y := y.left

  }

}
```

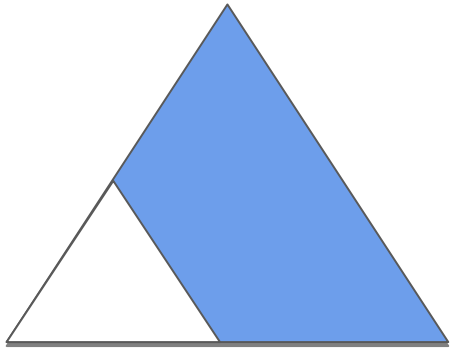


Tree(y) −∗ Tree(x)

Using magic wands

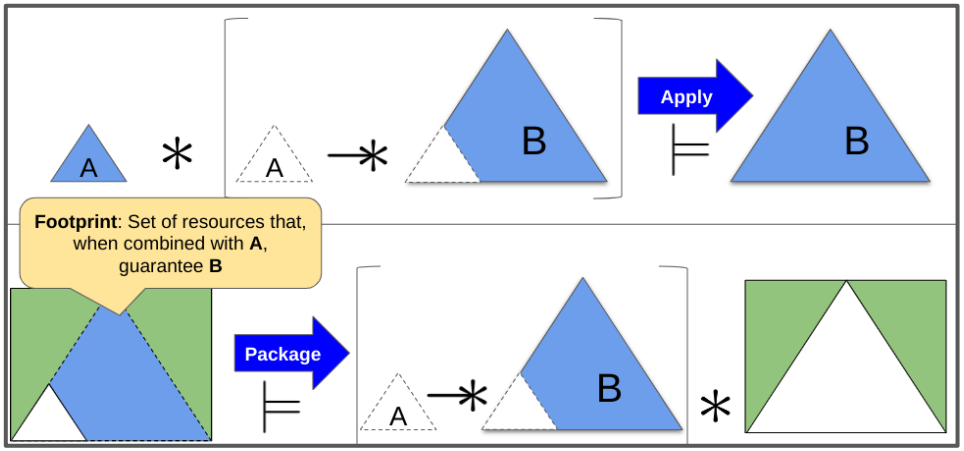



```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x
  package Tree(y) −* Tree(x)

  while(y.left != null)
  invariant Tree(y) * (Tree(y) −* Tree(x))
  {
    y := y.left

  }

}
```

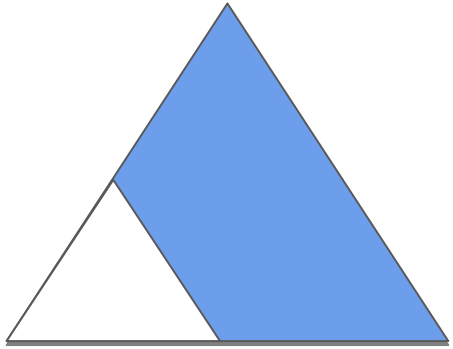


Tree(y) −* Tree(x)

Using magic wands

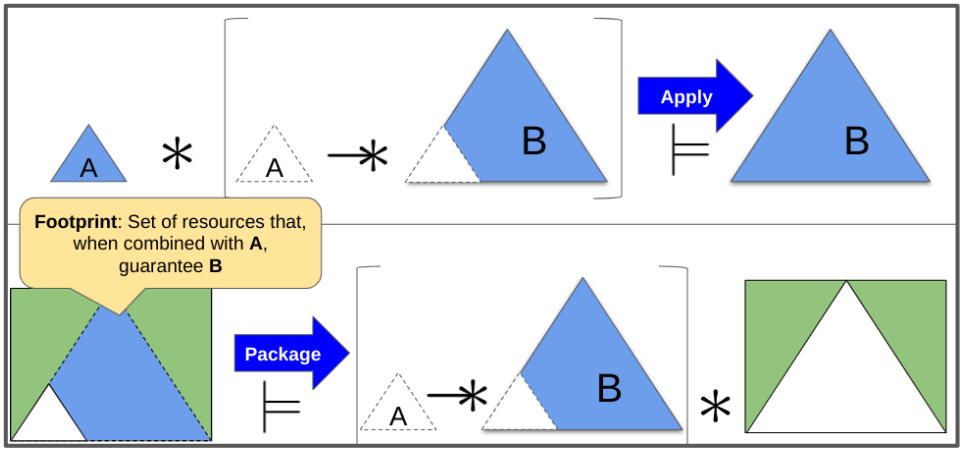



```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x
  package Tree(y) −∗ Tree(x)

  while(y.left != null)
  invariant Tree(y) ∗ (Tree(y) −∗ Tree(x))
  {
    y := y.left
    package Tree(y) −∗ Tree(x)
  }

}
```

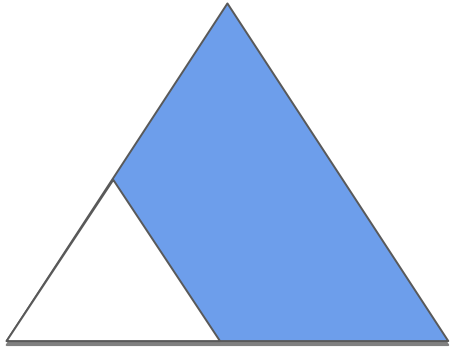


$$\mathrm{Tree}(y) \mathrel{-\!\!*} \mathrm{Tree}(x)$$

Using magic wands



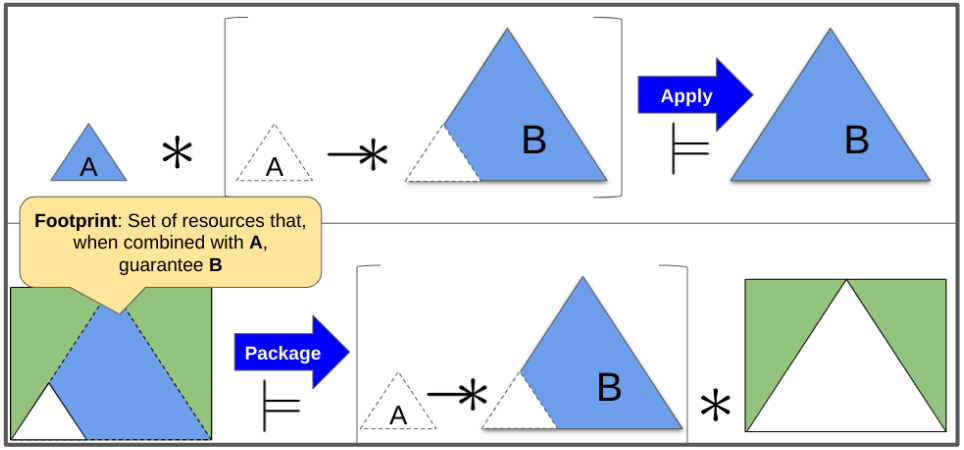

```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x
  package Tree(y) −∗ Tree(x)

  while(y.left != null)
  invariant Tree(y) ∗ (Tree(y) −∗ Tree(x))
  {
    y := y.left
    package Tree(y) −∗ Tree(x)
  }

}
```

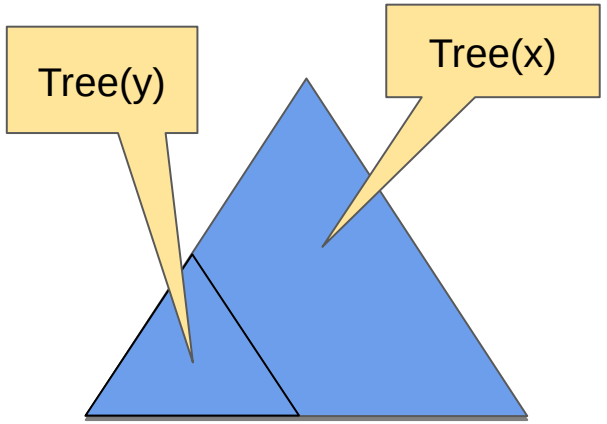


Tree(y) −∗ Tree(x)

Using magic wands

```
method leftLeaf(x: Ref) : (y: Ref)
  requires Tree(x)
  ensures Tree(x)
{
  y := x
  package Tree(y) —∗ Tree(x)

  while(y.left != null)
  invariant Tree(y) ∗ (Tree(y) —∗ Tree(x))
  {
    y := y.left
    package Tree(y) —∗ Tree(x)
  }
  apply Tree(y) —∗ Tree(x)
}
```





Challenge: Automatically compute a **small** and **valid** footprint

Challenge: Automatically compute a **small** and **valid** footprint

Not automated: The user must specify the footprint

STTT 2015

Witnessing the elimination of magic wands

Stefan Blom · Marieke Huisman

VerCors

Challenge: Automatically compute a **small** and **valid** footprint

Not automated: The user must specify the footprint

STTT 2015

Witnessing the elimination of magic wands

Stefan Blom · Marieke Huisman

VerCors

ECOOP 2015

Lightweight Support for Magic Wands in an Automatic Verifier

Malte Schwerhoff[1] and Alexander J. Summers[2]

Challenge: Automatically compute a **small** and **valid** footprint

Not automated: The user must specify the footprint

STTT 2015

Witnessing the elimination of magic wands

Stefan Blom · Marieke Huisman

VerCors

Automatically computes footprint

ECOOP 2015

Lightweight Support for Magic Wands in an Automatic Verifier

Malte Schwerhoff[1] and Alexander J. Summers[2]

VIPER

Challenge: Automatically compute a **small** and **valid** footprint

Not automated: The user must specify the footprint

STTT 2015

Witnessing the elimination of magic wands

Stefan Blom · Marieke Huisman

VerCors

Automatically computes footprint

Computed footprint sometimes invalid 

ECOOP 2015

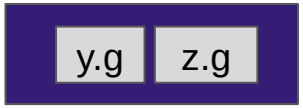
Lightweight Support for Magic Wands in an Automatic Verifier

Malte Schwerhoff[1] and Alexander J. Summers[2]

Examples of footprints

Examples of footprints

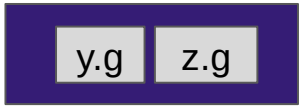
Current state



package $\mathrm{acc}(x.f) \mathrel{-\!\!*} (\mathrm{acc}(x.f) * \mathrm{acc}(y.g))$

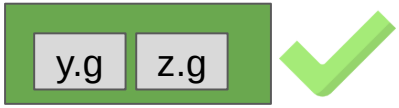
Examples of footprints

Current state



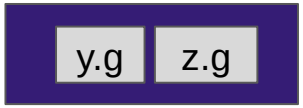
package acc(x.f) $\rightarrow\!\!*$ (acc(x.f) $*$ acc(y.g))

Valid footprints:



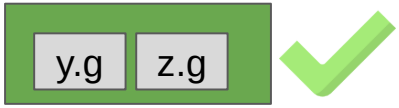
Examples of footprints

Current state



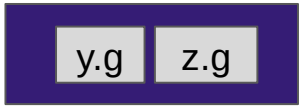
package acc(x.f) $\twoheadrightarrow$ (acc(x.f) $*$ acc(y.g))

Valid footprints:



Examples of footprints

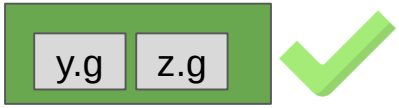
Current state



package acc(x.f) $-\!\!*$ (acc(x.f) $*$ acc(y.g))

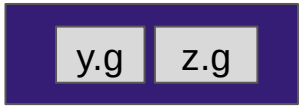
Smaller footprint

Valid footprints:



Examples of footprints

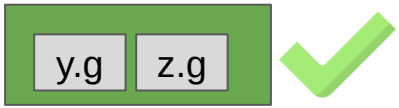
Current state



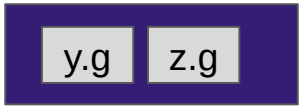
package $\mathrm{acc}(x.f) \twoheadrightarrow (\mathrm{acc}(x.f) * \mathrm{acc}(y.g))$

Smaller footprint

Valid footprints:

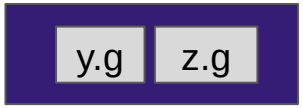


Current state



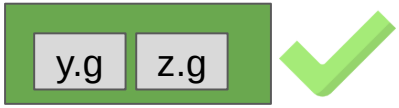
package $(\mathrm{acc}(x.f) * (x.f = y \vee x.f = z)) \twoheadrightarrow (\mathrm{acc}(x.f) * \mathrm{acc}(x.f.g))$

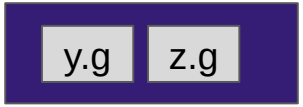
Examples of footprints

Current state 

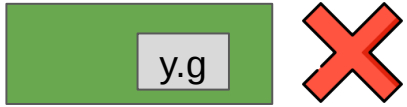
package acc(x.f) $-\!\!*$ (acc(x.f) $*$ acc(y.g))

Smaller footprint

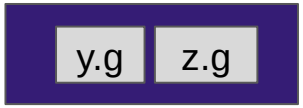
Valid footprints: 

Current state 

package (acc(x.f) $*$ (x.f = y $\vee$ x.f = z)) $-\!\!*$ (acc(x.f) $*$ acc(x.f.g))

Potential footprints: 

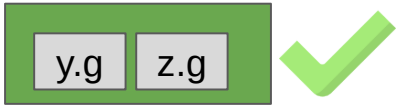
Examples of footprints

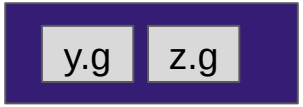
Current state 

package acc(x.f) $-\!\!*$ (acc(x.f) $*$ acc(y.g))

Smaller footprint

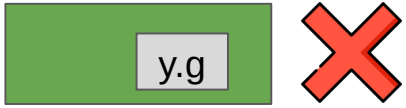
Valid footprints:

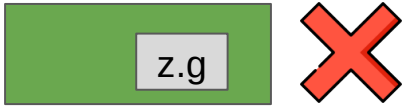


Current state 

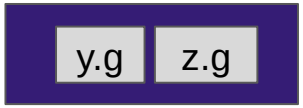
package (acc(x.f) $*$ (x.f = y $\vee$ x.f = z)) $-\!\!*$ (acc(x.f) $*$ acc(x.f.g))

Potential footprints:





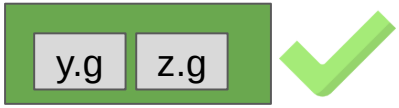
Examples of footprints

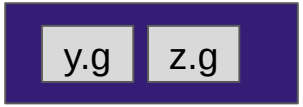
Current state 

package $acc(x.f) \twoheadrightarrow (acc(x.f) * acc(y.g))$

Smaller footprint

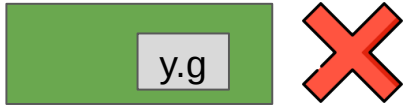
Valid footprints:

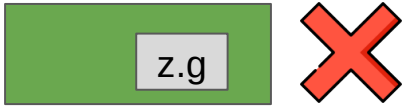


Current state 

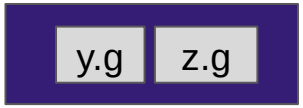
package $(acc(x.f) * (x.f = y \vee x.f = z)) \twoheadrightarrow (acc(x.f) * acc(x.f.g))$

Potential footprints:





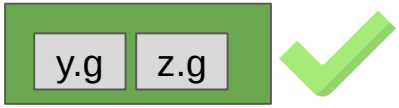
Examples of footprints

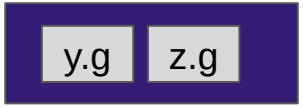
Current state 

package acc(x.f) $-\!\!*$ (acc(x.f) $*$ acc(y.g))

Smaller footprint

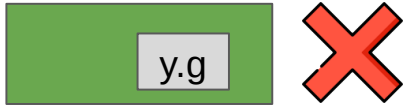
Valid footprints:

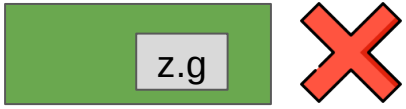


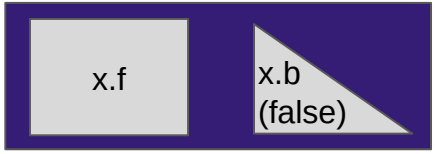
Current state 

package (acc(x.f) $*$ (x.f = y $\vee$ x.f = z)) $-\!\!*$ (acc(x.f) $*$ acc(x.f.g))

Potential footprints:



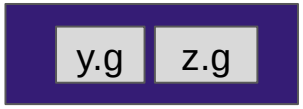


Current state 

package acc(x.b, 1/2) $-\!\!*$ (acc(x.b, 1/2) $*$ (x.b $\Rightarrow$ acc(x.f)))

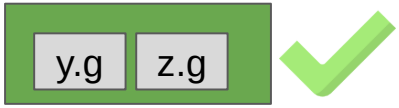
Valid footprints:

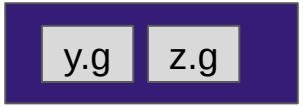
Examples of footprints

Current state 

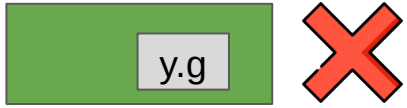
package $acc(x.f) \mathrel{-\!\!*} (acc(x.f) * acc(y.g))$

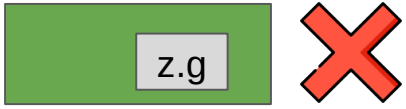
Smaller footprint

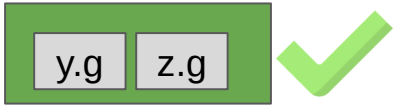
Valid footprints: 

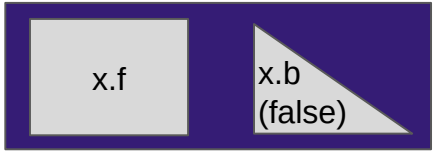
Current state 

package $(acc(x.f) * (x.f = y \vee x.f = z)) \mathrel{-\!\!*} (acc(x.f) * acc(x.f.g))$

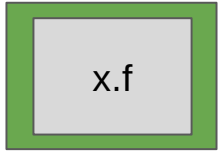
Potential footprints: 



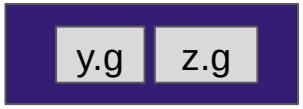


Current state 

package $acc(x.b, 1/2) \mathrel{-\!\!*} (acc(x.b, 1/2) * (x.b \Rightarrow acc(x.f)))$

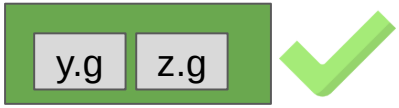
Valid footprints: 

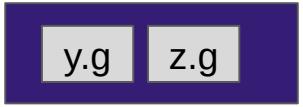
Examples of footprints

Current state 

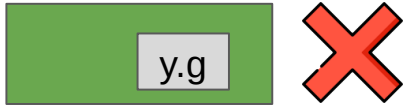
package $acc(x.f) \mathrel{-\!\!*} (acc(x.f) * acc(y.g))$

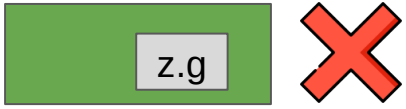
Smaller footprint

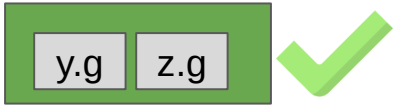
Valid footprints: 

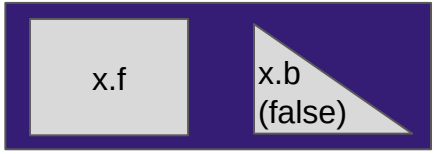
Current state 

package $(acc(x.f) * (x.f = y \vee x.f = z)) \mathrel{-\!\!*} (acc(x.f) * acc(x.f.g))$

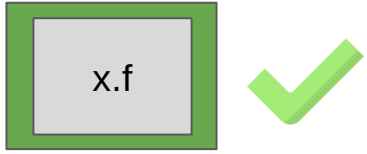
Potential footprints: 

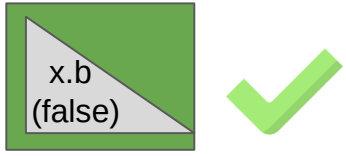




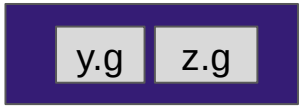
Current state 

package $acc(x.b, 1/2) \mathrel{-\!\!*} (acc(x.b, 1/2) * (x.b \Rightarrow acc(x.f)))$

Valid footprints: 

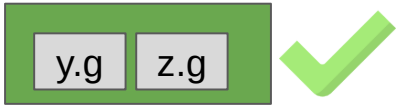


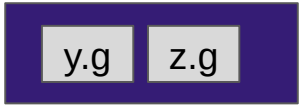
Examples of footprints

Current state 

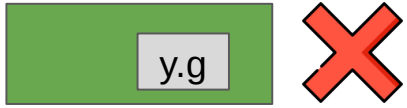
package $acc(x.f) \mathrel{-\!\!*} (acc(x.f) * acc(y.g))$

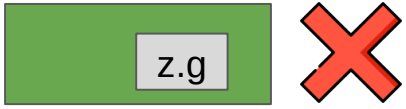
Smaller footprint

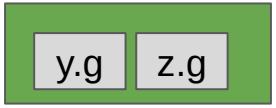
Valid footprints: 

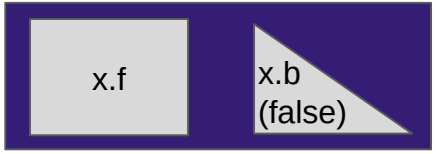
Current state 

package $(acc(x.f) * (x.f = y \vee x.f = z)) \mathrel{-\!\!*} (acc(x.f) * acc(x.f.g))$

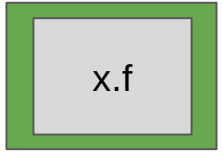
Potential footprints: 

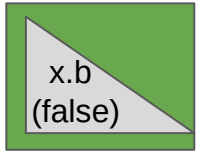




Current state 

package $acc(x.b, 1/2) \mathrel{-\!\!*} (acc(x.b, 1/2) * (x.b \Rightarrow acc(x.f)))$

Valid footprints: 



No minimal footprint → no “best” algorithm!

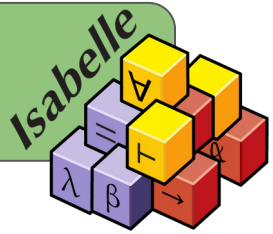
Contributions

1. Package logic

Contributions

Parametric logical framework for computing footprints
Sound and complete

1. Package logic

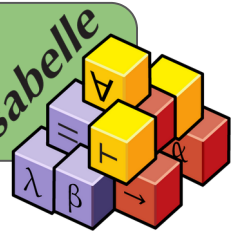


Contributions

Standard wand

Parametric logical framework for computing footprints
Sound and complete

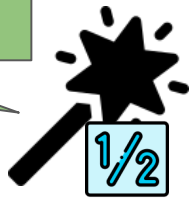
1. Package logic



Contributions

Standard wand

2. Combinable wand
Novel definition of the magic wand, with useful properties when combined with fractional permissions



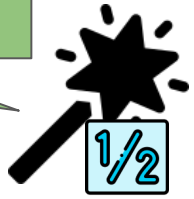
Parametric logical framework for computing footprints
Sound and complete

1. Package logic

Contributions

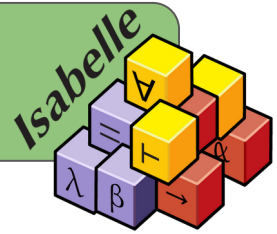
Standard wand

2. Combinable wand
Novel definition of the magic wand, with useful properties when combined with fractional permissions



Parametric logical framework for computing footprints
Sound and complete

1. Package logic



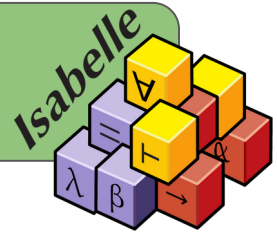
3. Implementations and evaluation

Contributions

Standard wand

Parametric logical framework for computing footprints
Sound and complete

1. Package logic



2. Combinable wand
Novel definition of the magic wand, with useful properties when combined with fractional permissions

3. Implementations and evaluation

Computing a footprint using the package logic (simplified)

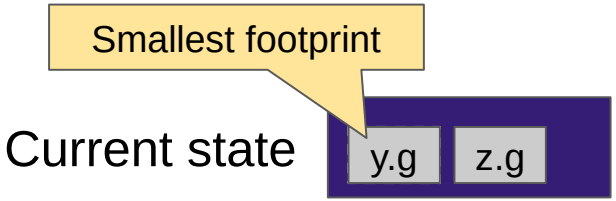
$$\textbf{package}\ \mathrm{acc}(x.f) \mathrel{-\!\!*} \mathrm{acc}(x.f) \ * \ \mathrm{acc}(y.g)$$

Current state

Footprint: Set of resources that, when combined with acc(x.f)**,** guarantee acc(x.f) * acc(y.g)

Computing a footprint using the package logic (simplified)

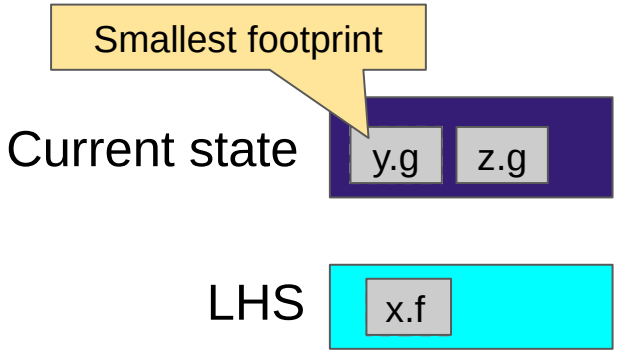
$$\textbf{package}\ \mathrm{acc}(x.f) \mathrel{-\!\!*} \mathrm{acc}(x.f) \ * \ \mathrm{acc}(y.g)$$



Footprint: Set of resources that, when combined with acc(x.f)**,** guarantee acc(x.f) * acc(y.g)

Computing a footprint using the package logic (simplified)

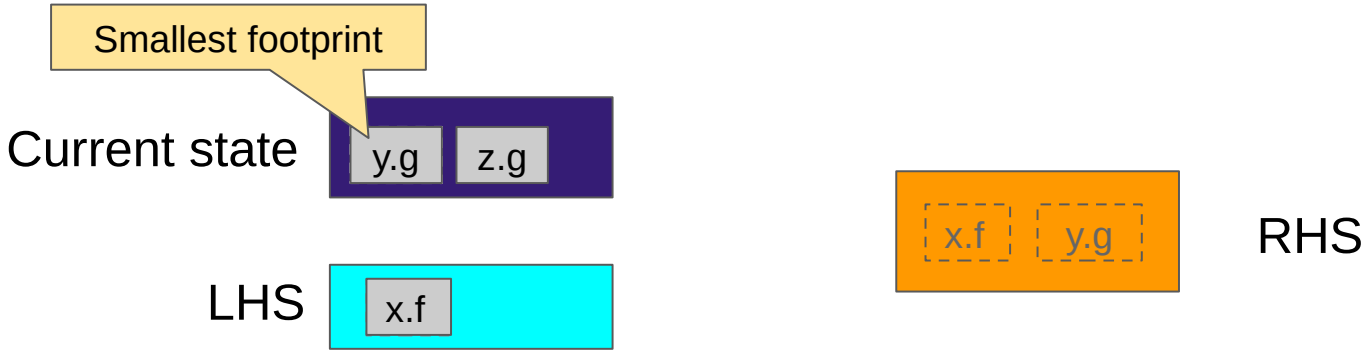
package acc(x.f) $\multimap$ acc(x.f) $*$ acc(y.g)



Footprint: Set of resources that, when combined with acc(x.f)**,** guarantee acc(x.f) $*$ acc(y.g)

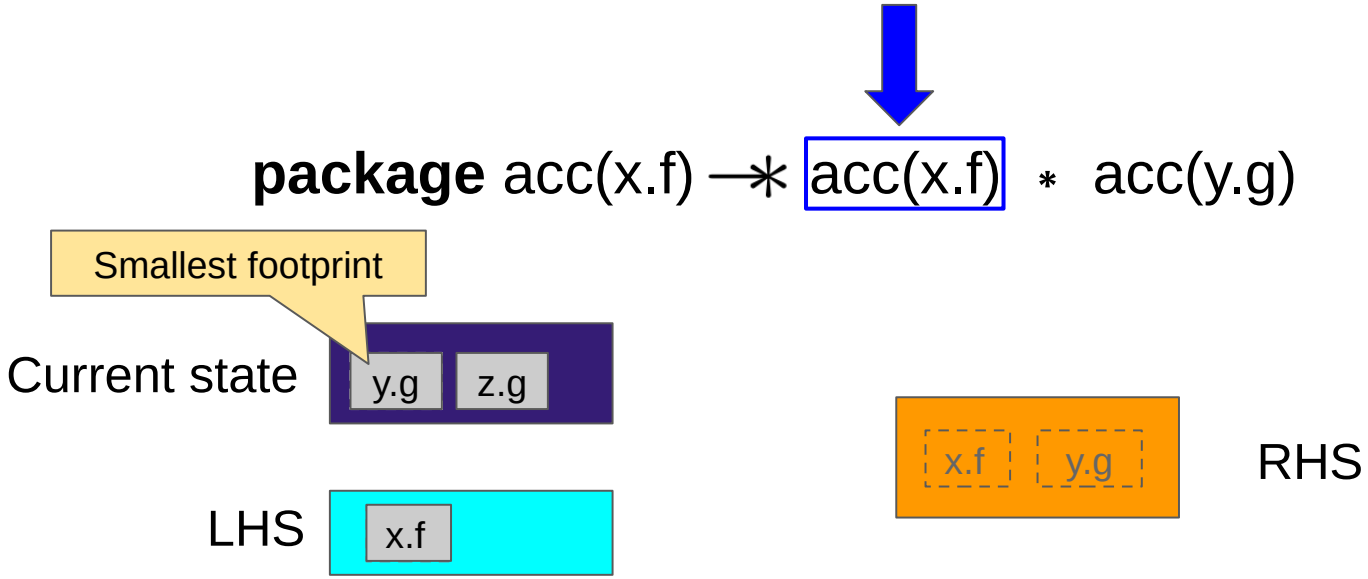
Computing a footprint using the package logic (simplified)

package acc(x.f) $\rightarrow\!\!*$ acc(x.f) $*$ acc(y.g)



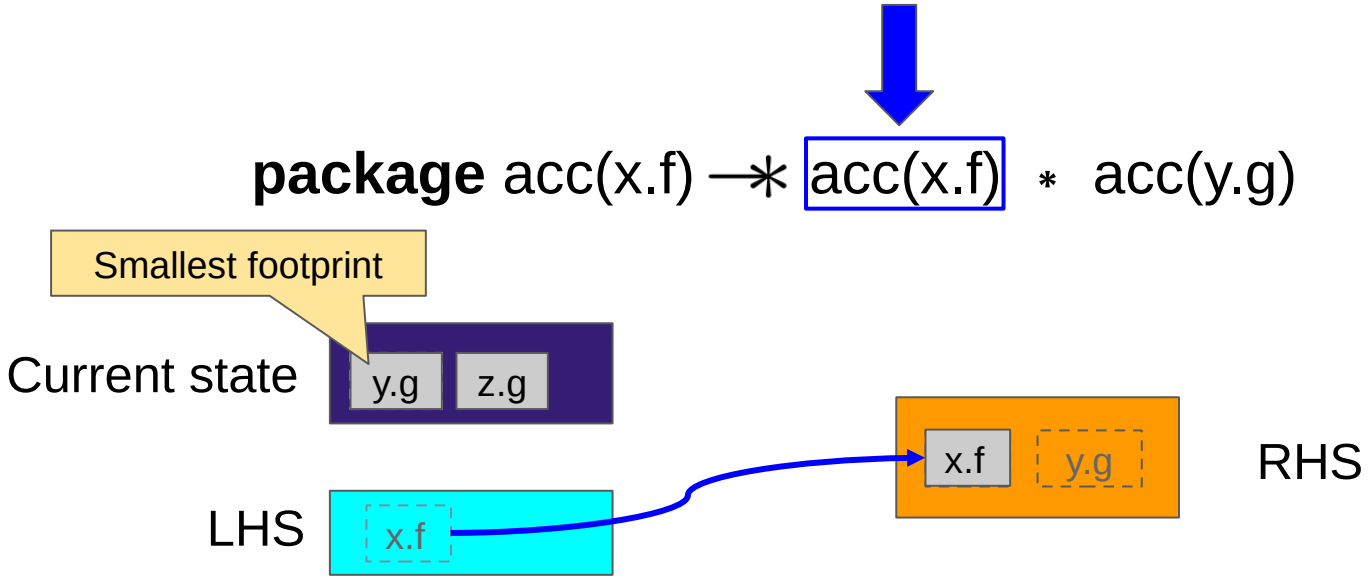
Footprint: Set of resources that, when combined with acc(x.f)**,** guarantee acc(x.f) $*$ acc(y.g)

Computing a footprint using the package logic (simplified)



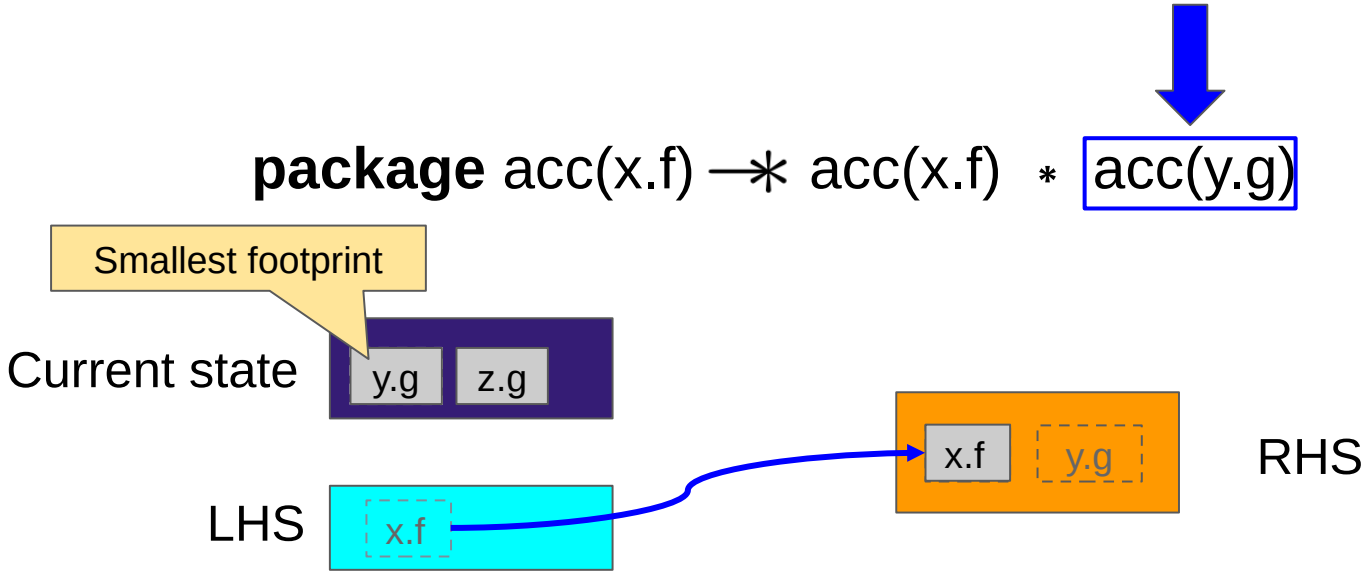
Footprint: Set of resources that, when combined with acc(x.f)**,** guarantee acc(x.f) ∗ acc(y.g)

Computing a footprint using the package logic (simplified)



Footprint: Set of resources that, when combined with acc(x.f)**,** guarantee acc(x.f) ∗ acc(y.g)

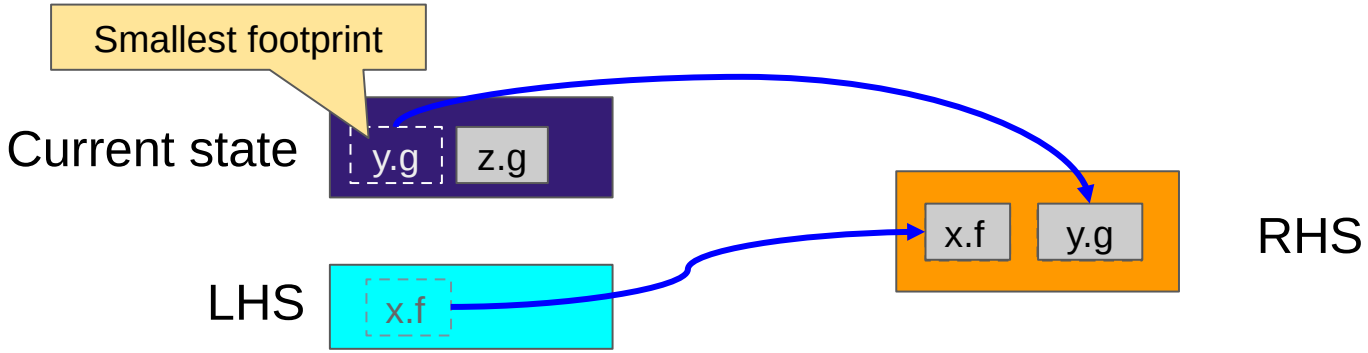
Computing a footprint using the package logic (simplified)



Footprint: Set of resources that, when combined with acc(x.f)**,** guarantee acc(x.f) ∗ acc(y.g)

Computing a footprint using the package logic (simplified)

package acc(x.f) $\rightarrow\!\!*$ acc(x.f) $*$ acc(y.g)



Footprint: Set of resources that, when combined with acc(x.f)**,** guarantee acc(x.f) $*$ acc(y.g)

Using the package logic

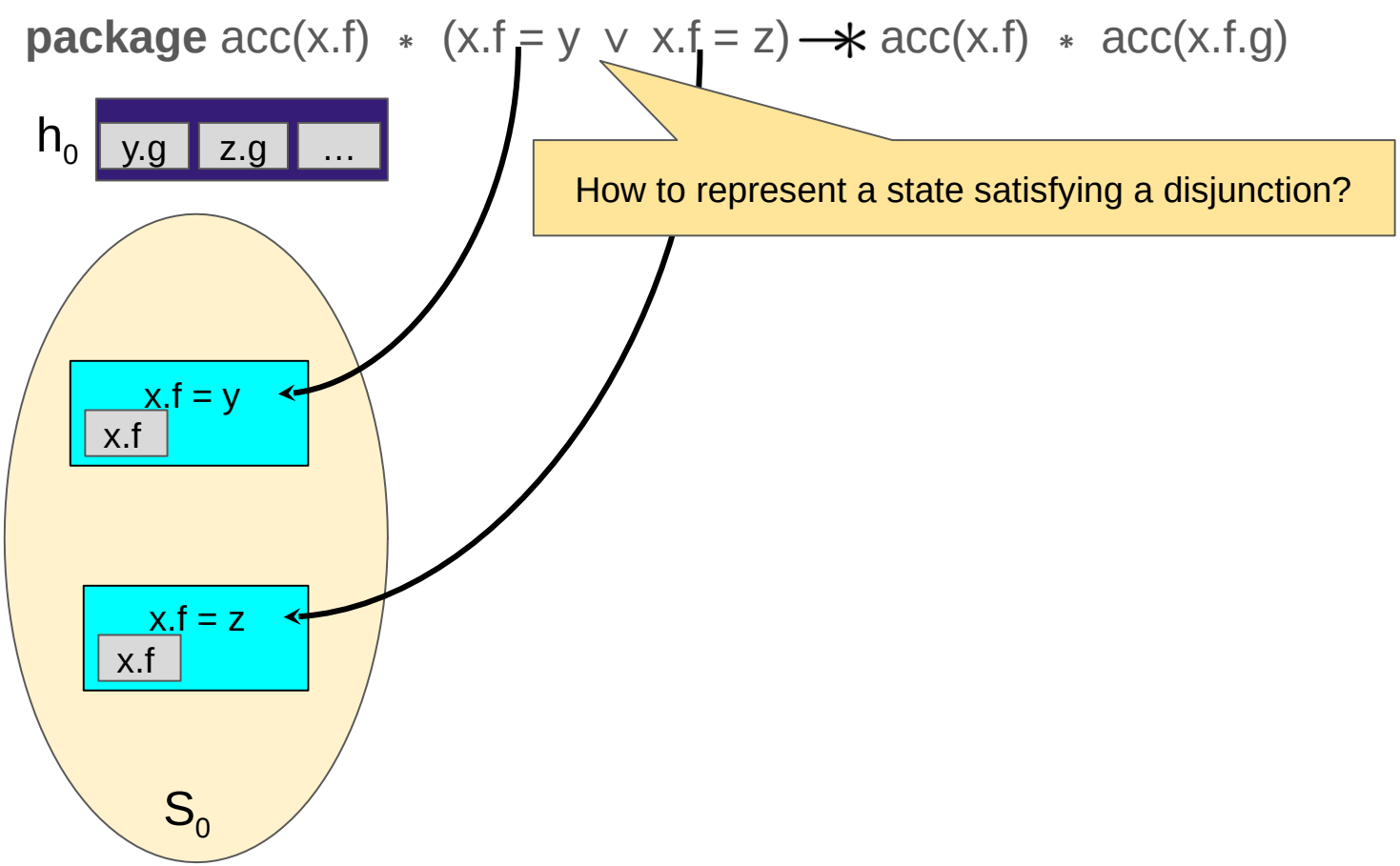
package $\mathrm{acc}(x.f) \;*\; (x.f = y \;\vee\; x.f = z) \mathrel{-\!\!*} \mathrm{acc}(x.f) \;*\; \mathrm{acc}(x.f.g)$

Using the package logic

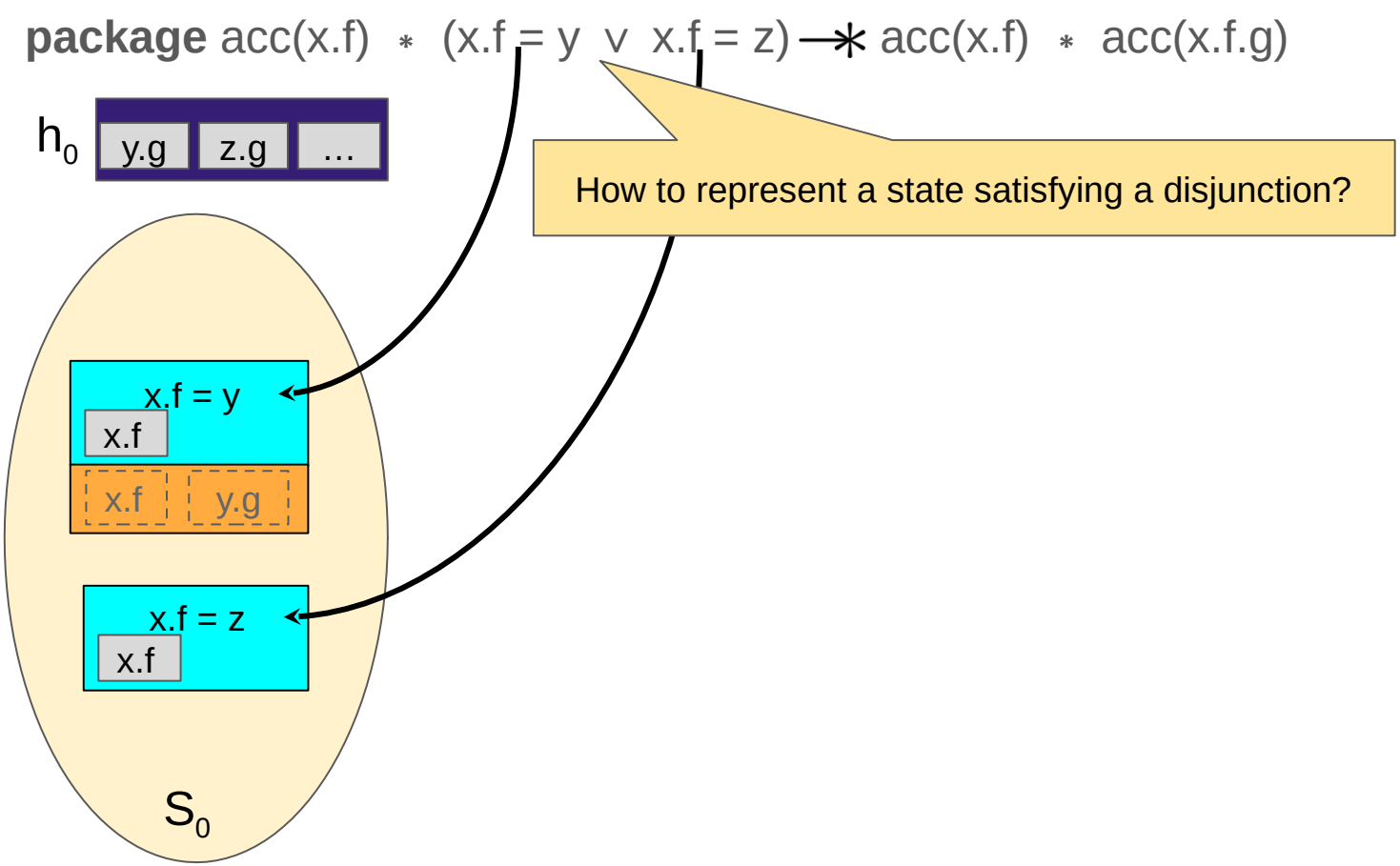
package acc(x.f) ∗ (x.f = y ∨ x.f = z) −∗ acc(x.f) ∗ acc(x.f.g)

How to represent a state satisfying a disjunction?

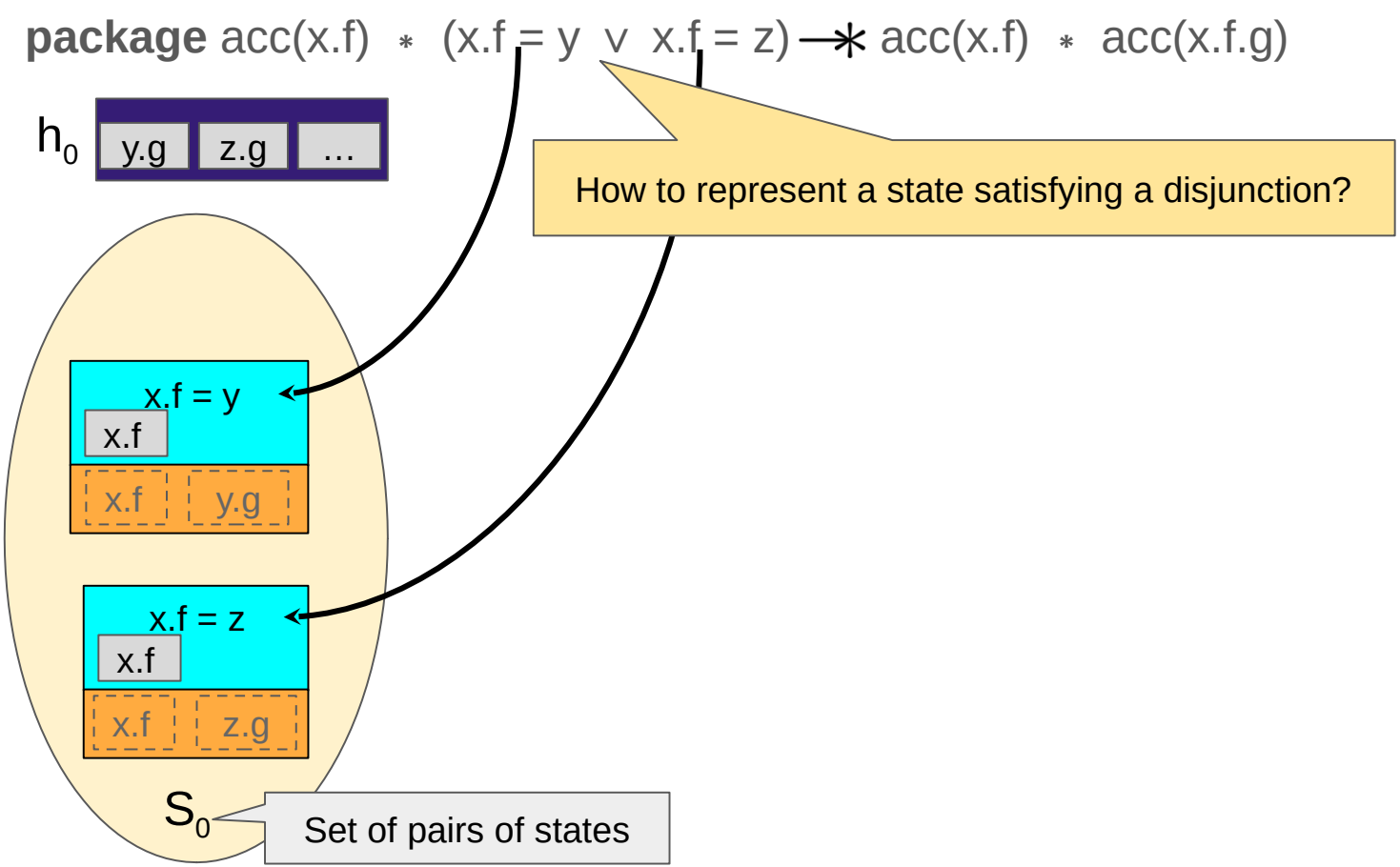
Using the package logic



Using the package logic

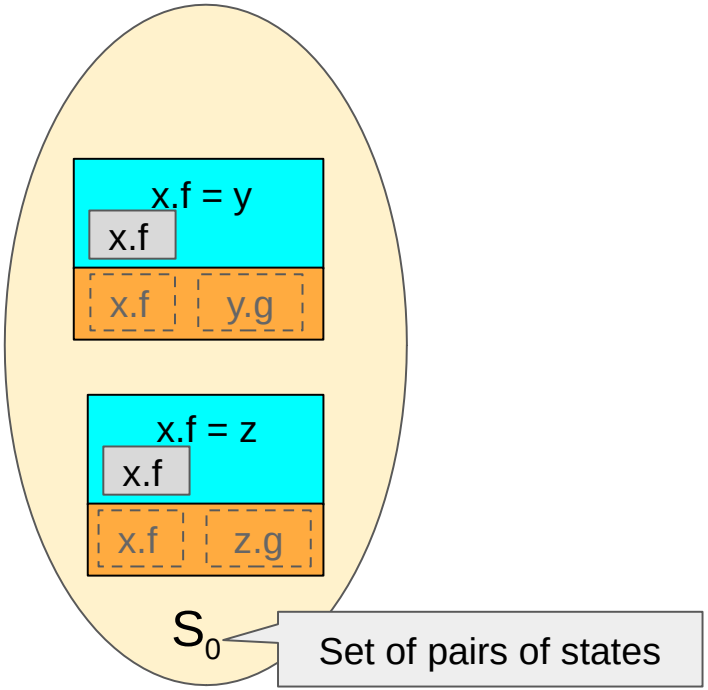


Using the package logic

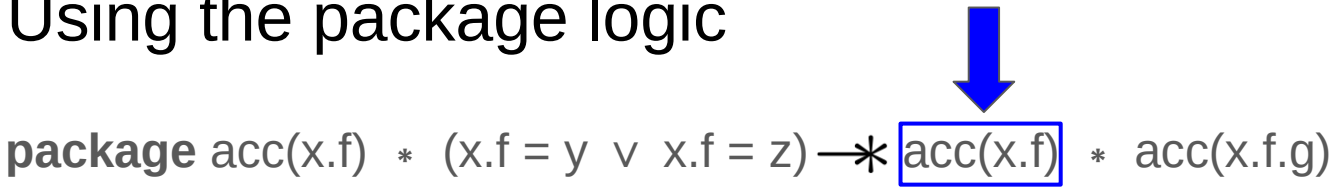


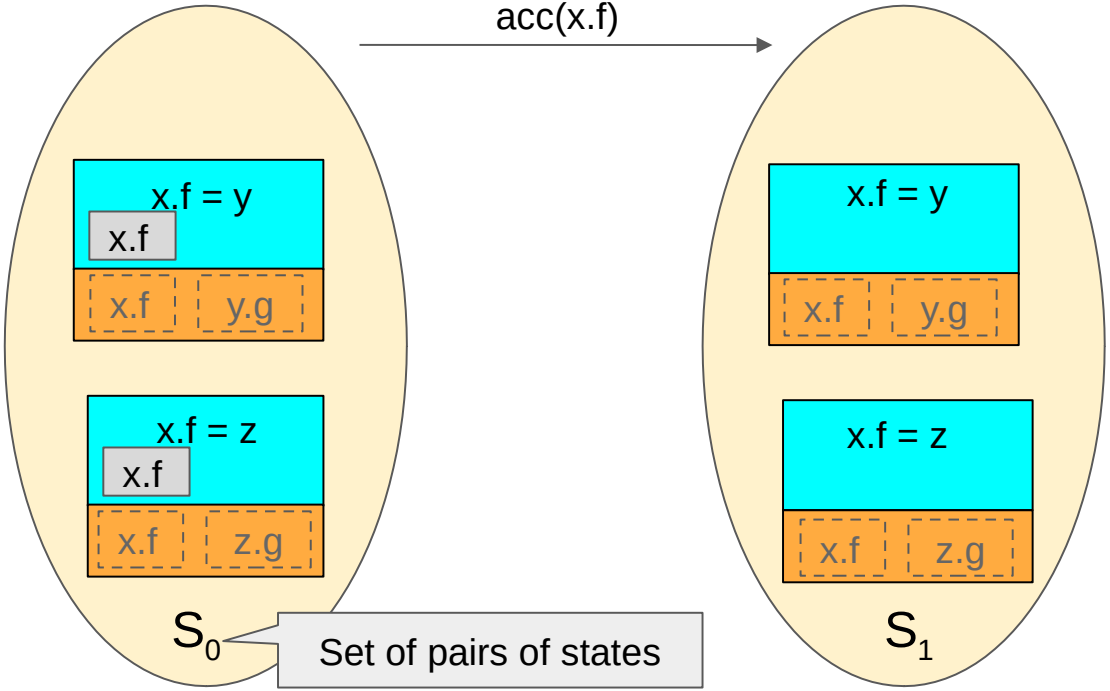
Using the package logic

package acc(x.f) ∗ (x.f = y ∨ x.f = z) −∗ acc(x.f) ∗ acc(x.f.g)



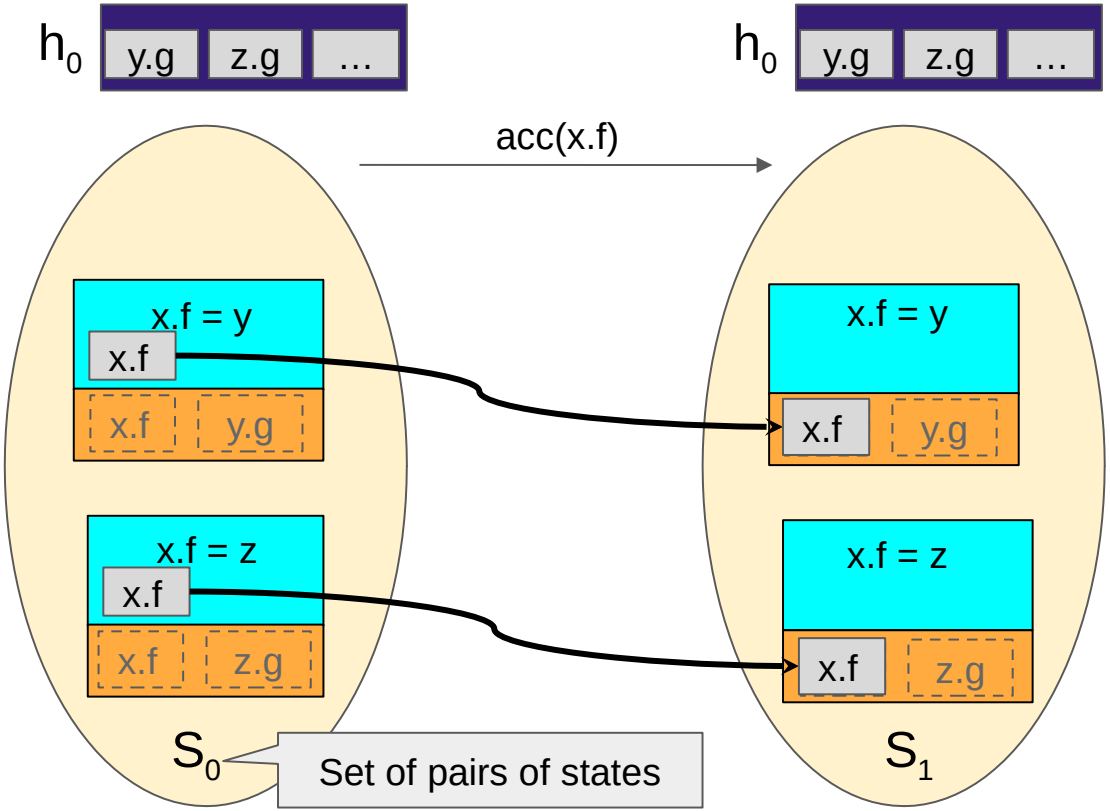
Using the package logic



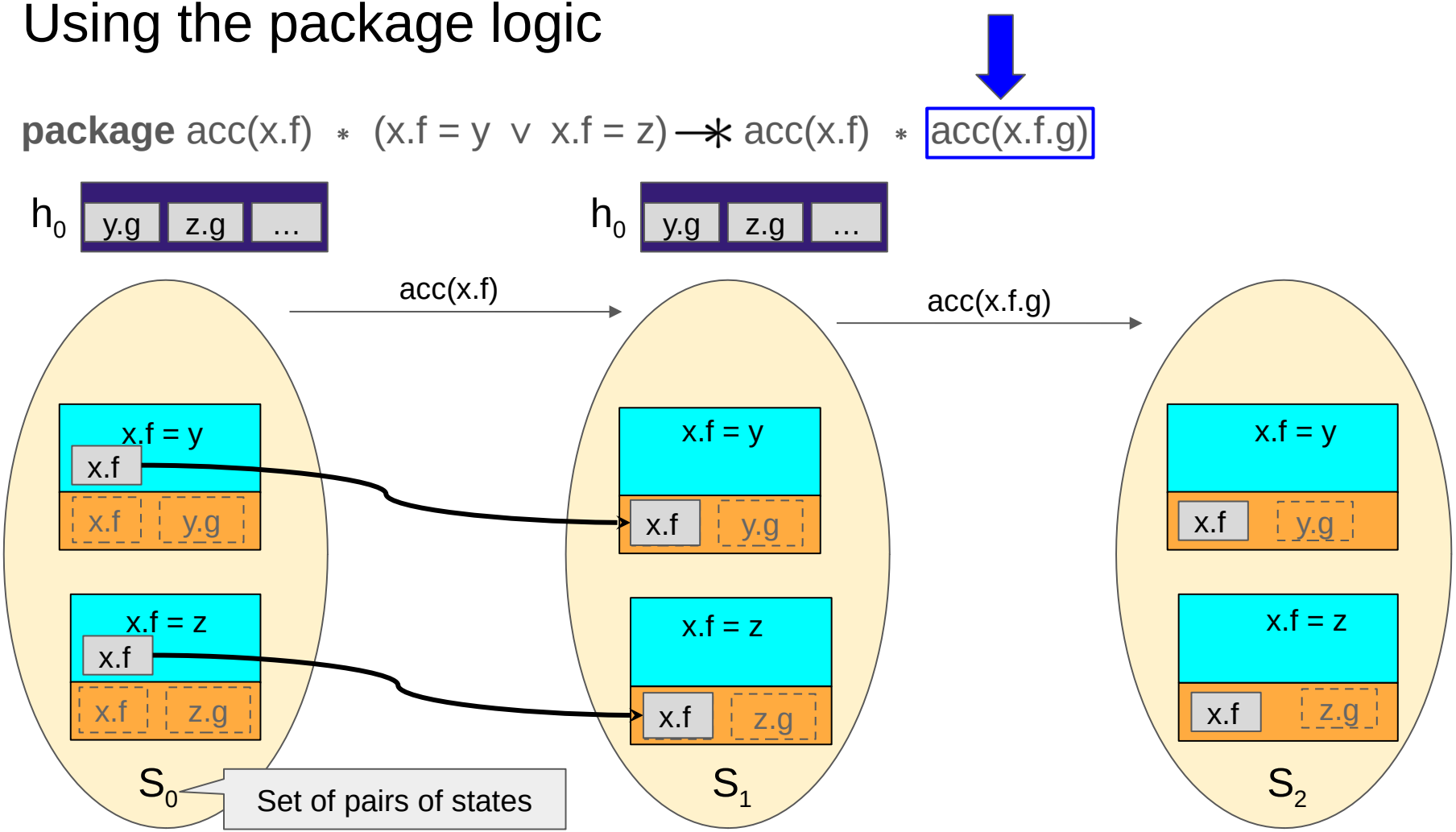


Using the package logic

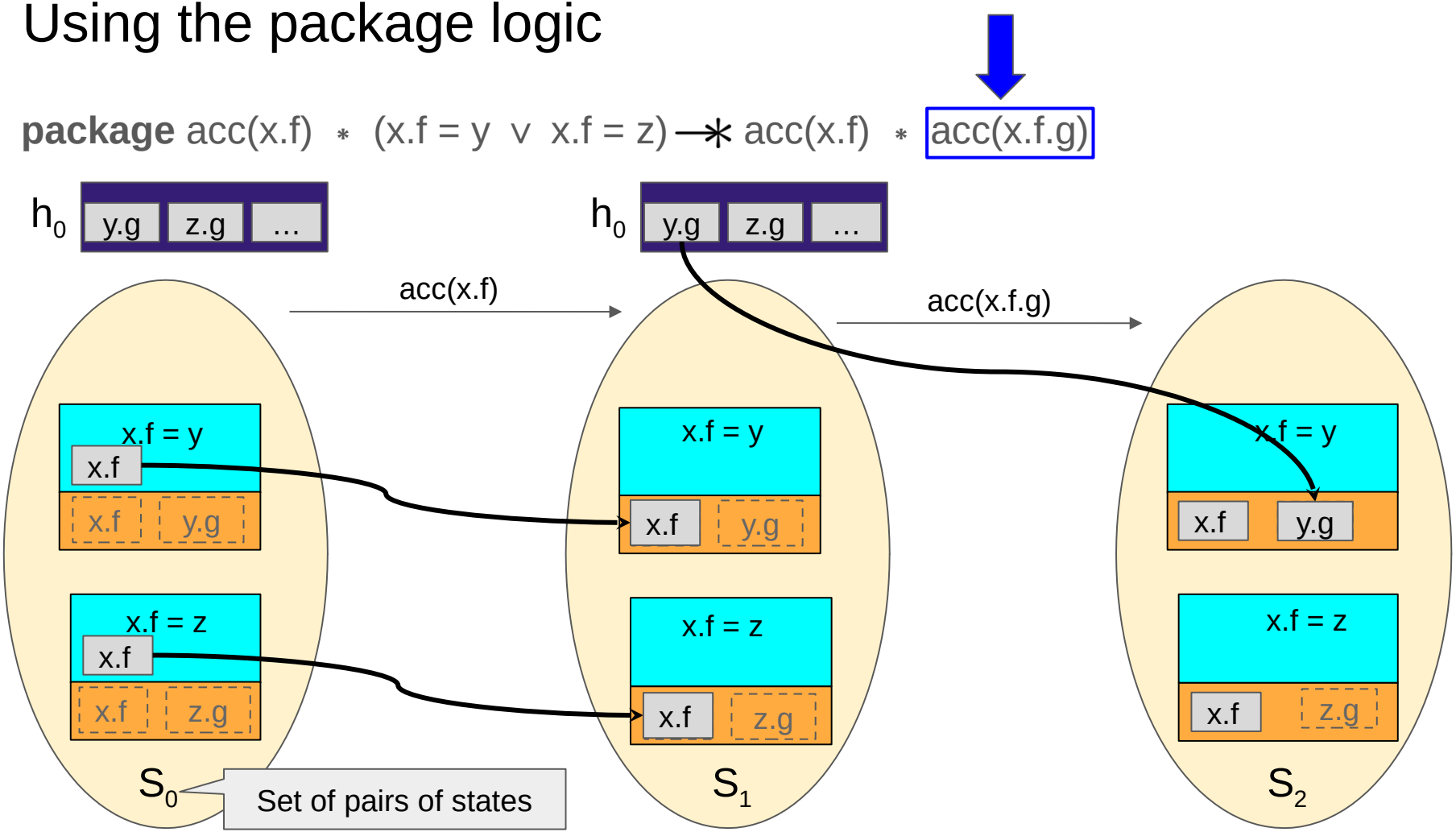
package acc(x.f) ∗ (x.f = y ∨ x.f = z) −∗ acc(x.f) ∗ acc(x.f.g)



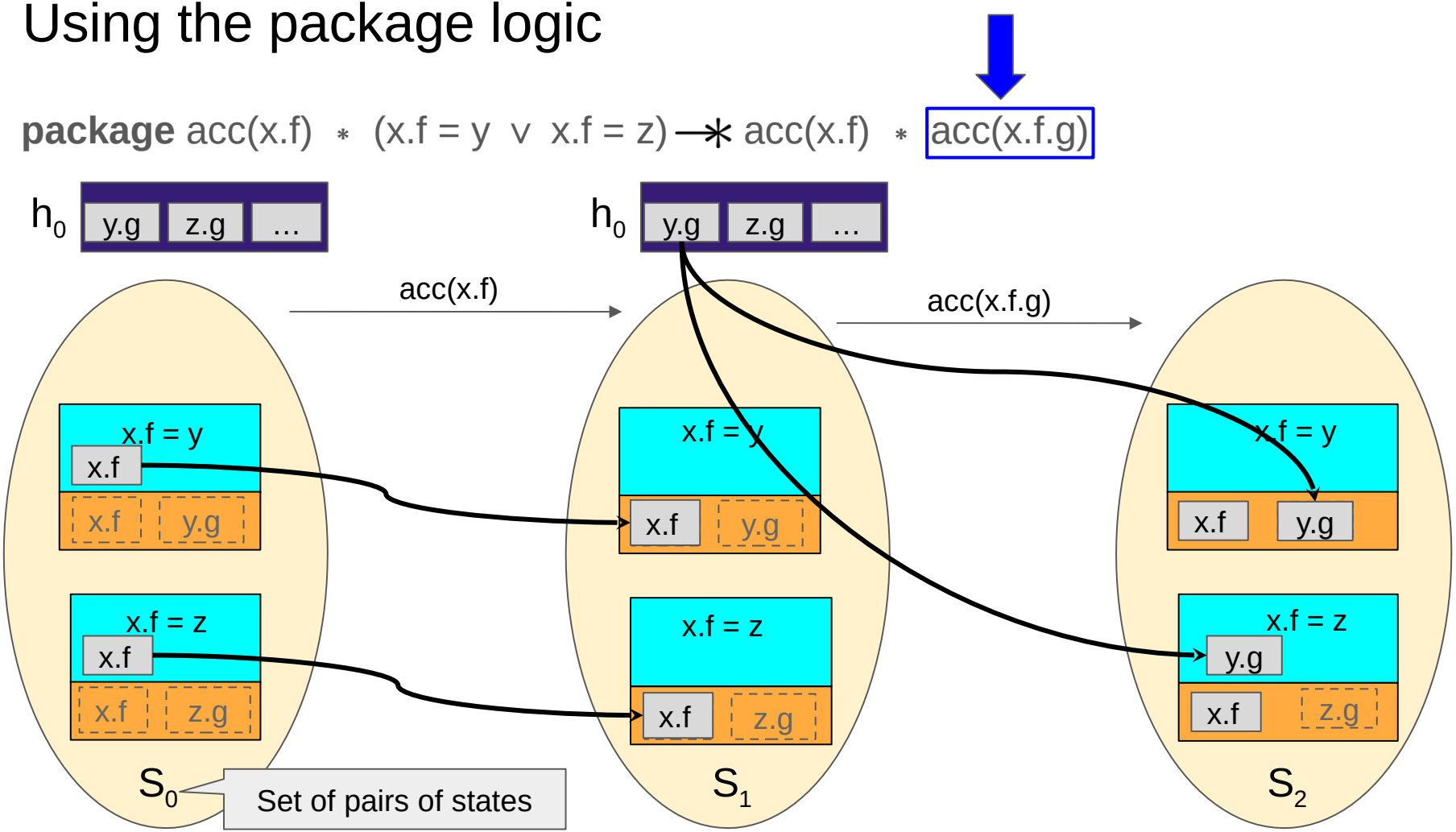
Using the package logic



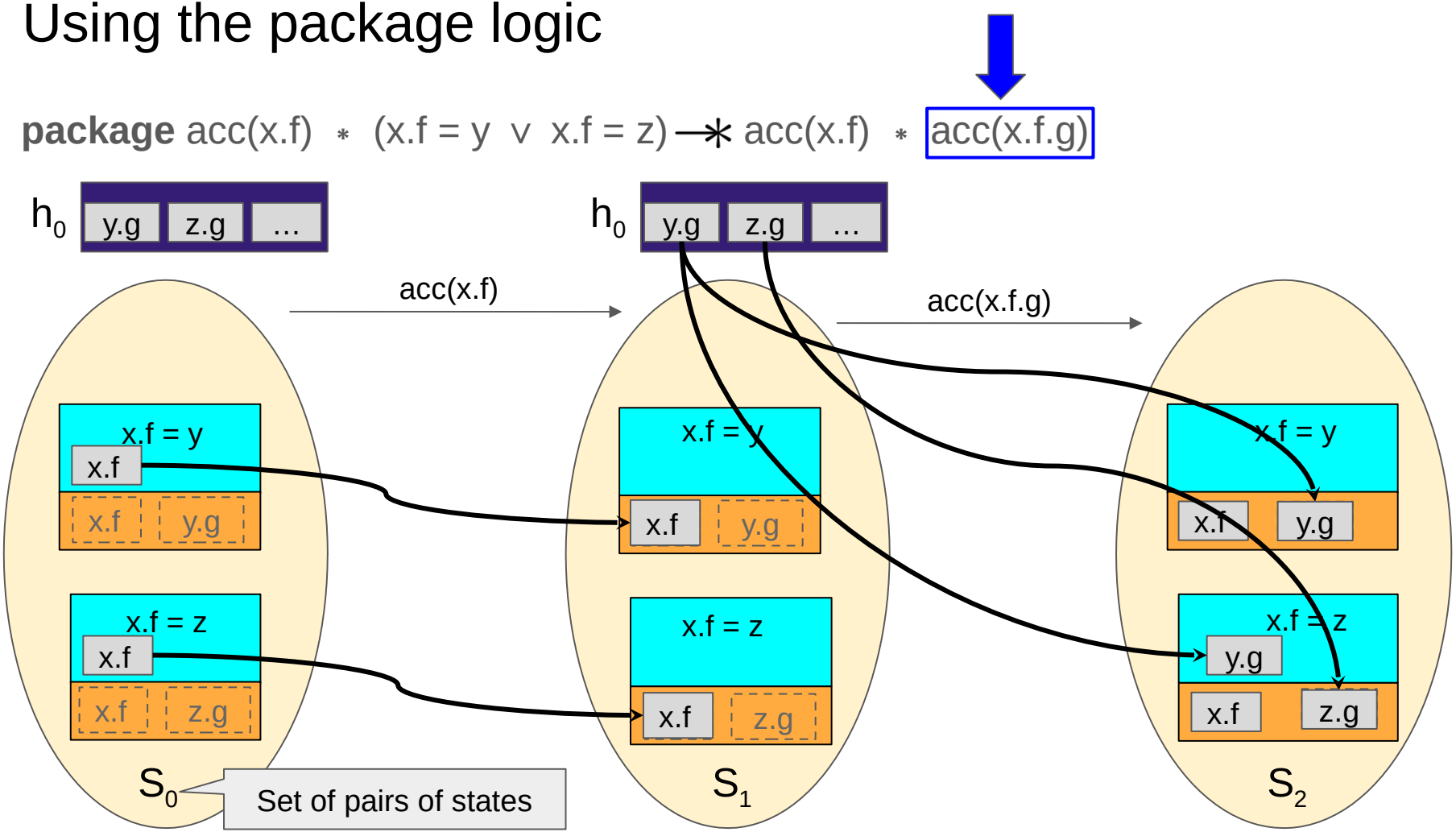
Using the package logic



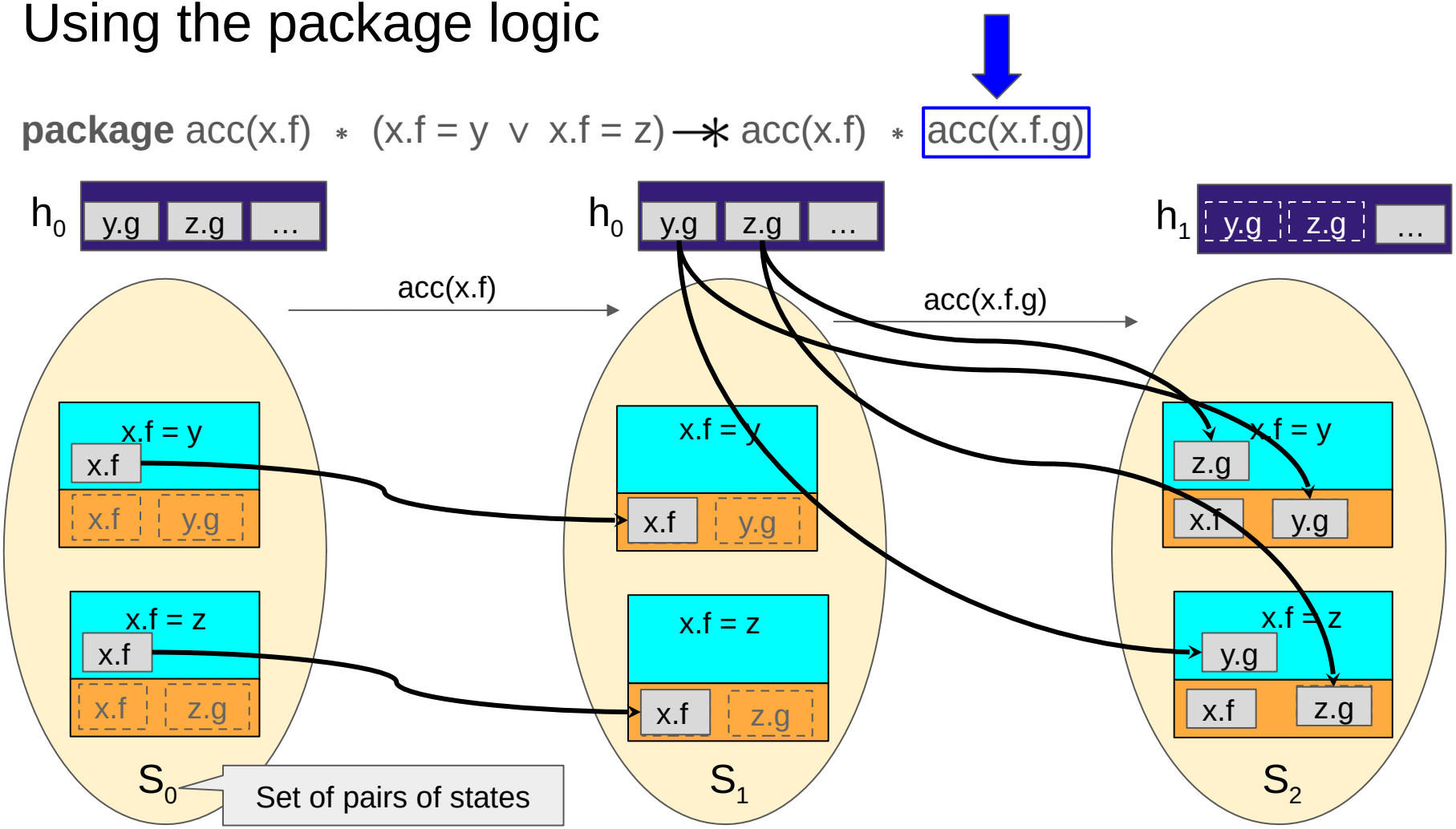
Using the package logic



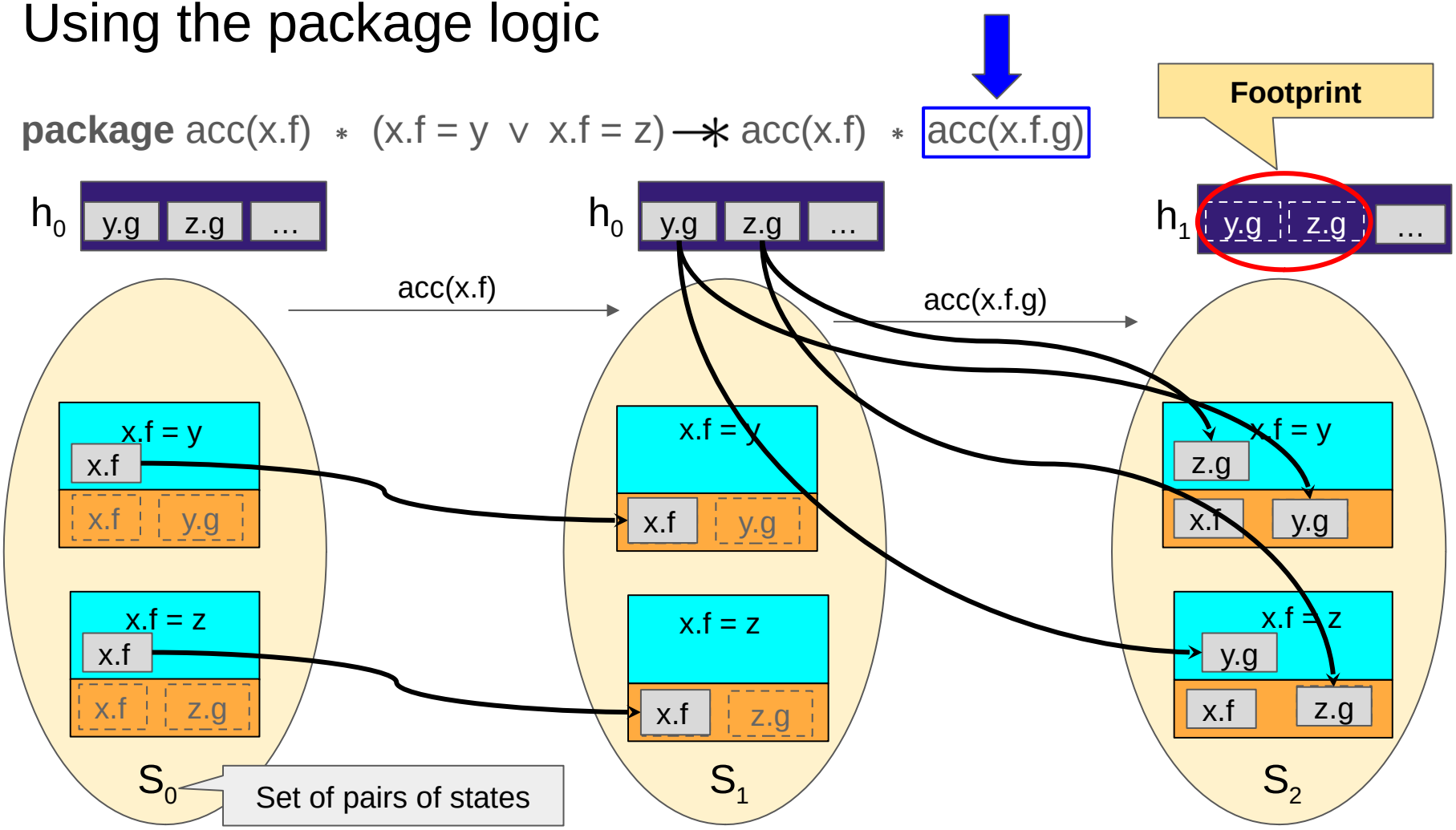
Using the package logic



Using the package logic

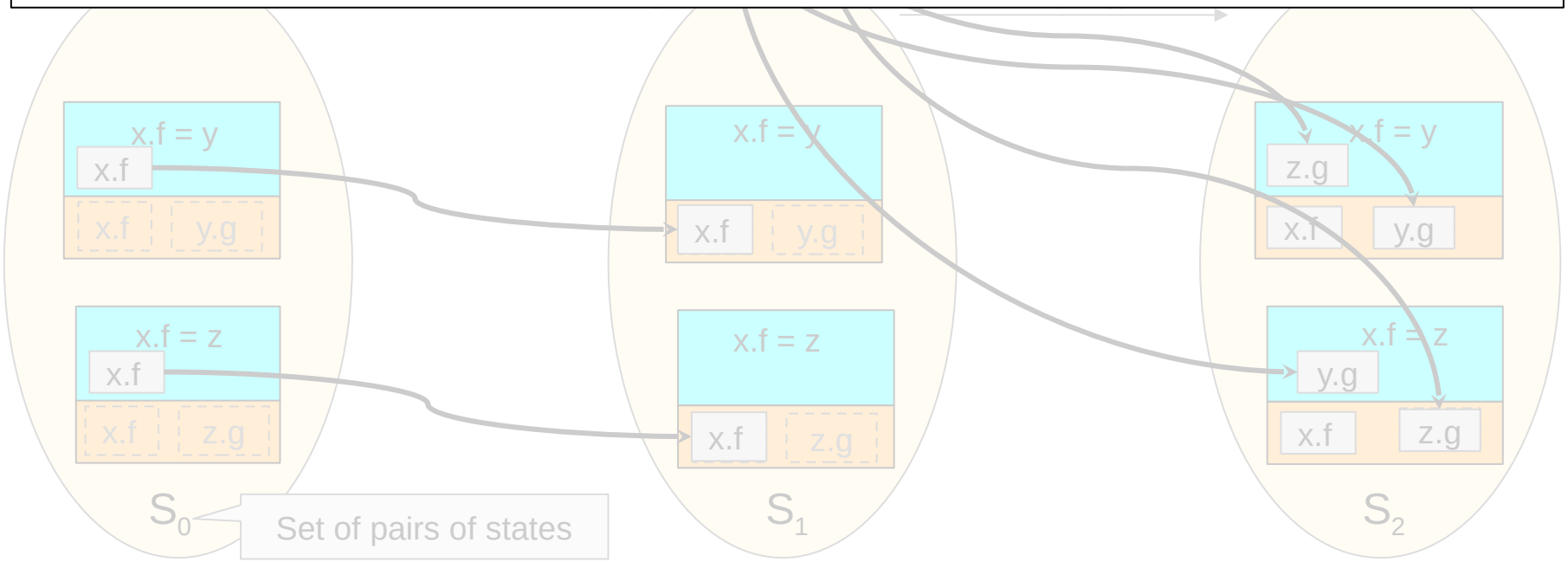


Using the package logic



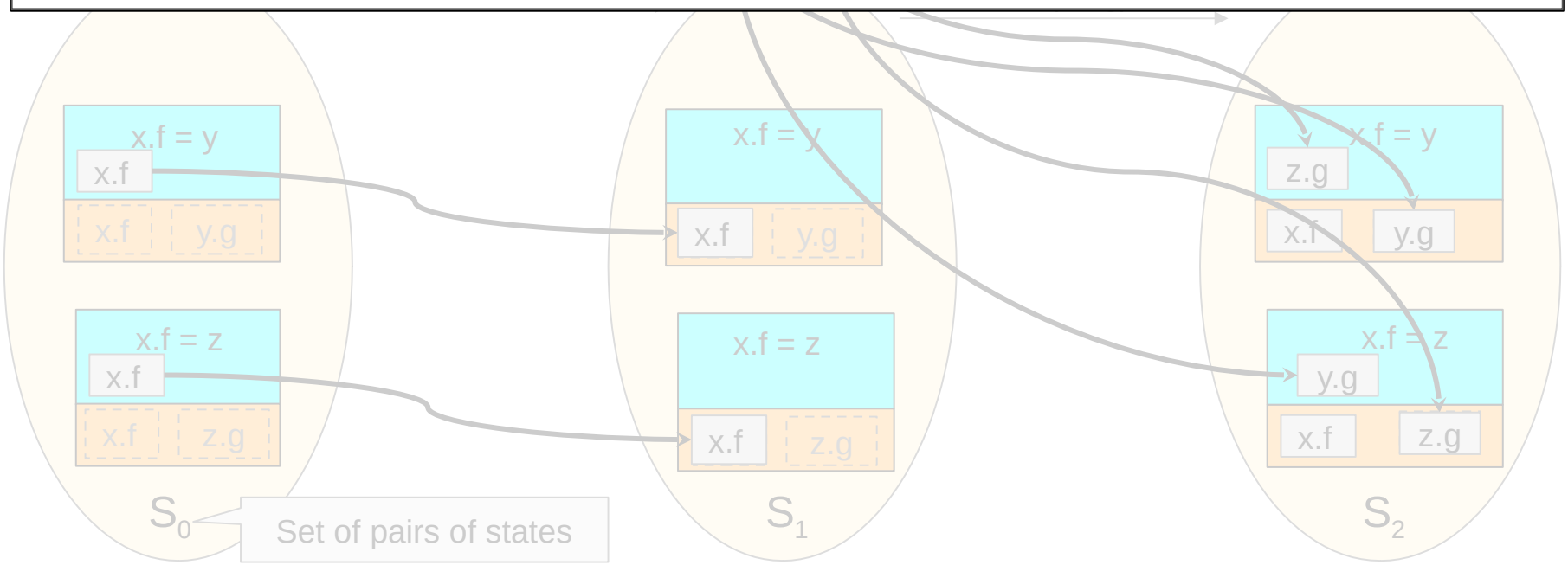
Using the package logic

$$\dfrac{\dfrac{\cdots}{\langle \texttt{acc(x.f)}, (h_0, S_0)\rangle \rightsquigarrow (h_0, S_1)}\,Atom \qquad \dfrac{\dfrac{\cdots}{\langle \texttt{acc(x.f.g)}, (h_1, S')\rangle \rightsquigarrow (h_1, S_2)}\,Atom}{\langle \texttt{acc(x.f.g)}, (h_0, S_1)\rangle \rightsquigarrow (h_1, S_2)}\,\dagger\,Extract}{\langle \texttt{acc(x.f) * acc(x.f.g)}, (h_0, S_0)\rangle \rightsquigarrow (h_1, S_2)}\,Star$$



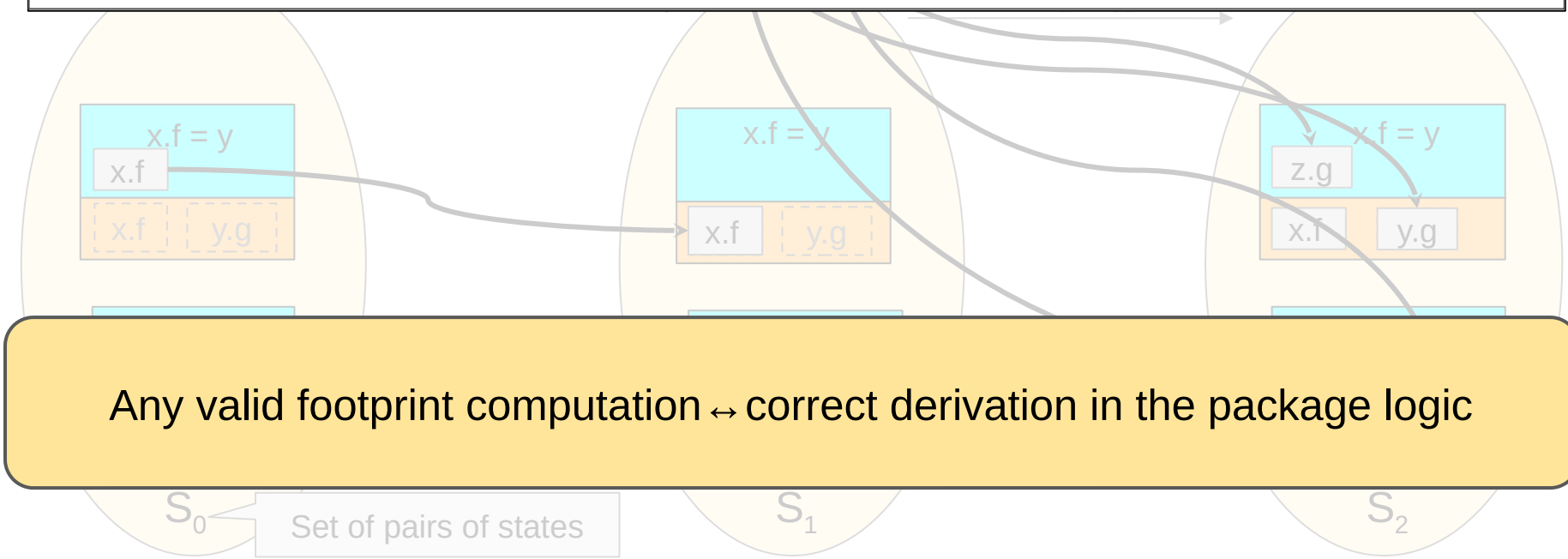
Using the package logic

$$\dfrac{\dfrac{\dots}{\langle \texttt{acc(x.f)}, (h_0, S_0)\rangle \rightsquigarrow (h_0, S_1)}\,Atom \qquad \dfrac{\dfrac{\dots}{\langle \texttt{acc(x.f.g)}, (h_1, S')\rangle \rightsquigarrow (h_1, S_2)}\,Atom}{\langle \texttt{acc(x.f.g)}, (h_0, S_1)\rangle \rightsquigarrow (h_1, S_2)}\,\dagger\,Extract}{\langle \texttt{acc(x.f) * acc(x.f.g)}, (h_0, S_0)\rangle \rightsquigarrow (h_1, S_2)}\,Star$$



Using the package logic

$$\dfrac{\dfrac{\dots}{\langle \texttt{acc(x.f)}, (h_0, S_0)\rangle \rightsquigarrow (h_0, S_1)}\,Atom \qquad \dfrac{\dfrac{\dots}{\langle \texttt{acc(x.f.g)}, (h_1, S')\rangle \rightsquigarrow (h_1, S_2)}\,Atom}{\langle \texttt{acc(x.f.g)}, (h_0, S_1)\rangle \rightsquigarrow (h_1, S_2)}\,\dagger\,Extract}{\langle \texttt{acc(x.f) * acc(x.f.g)}, (h_0, S_0)\rangle \rightsquigarrow (h_1, S_2)}\,Star$$

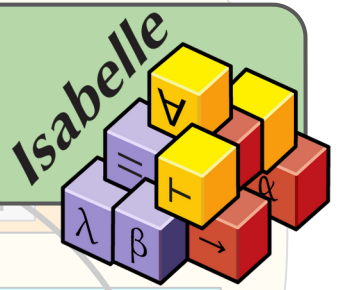


Any valid footprint computation ↔ correct derivation in the package logic

Using the package logic

$$\dfrac{\dfrac{\dots}{\langle \texttt{acc(x.f)}, (h_0, S_0)\rangle \rightsquigarrow (h_0, S_1)}\,Atom \qquad \dfrac{\dfrac{\dots}{\langle \texttt{acc(x.f.g)}, (h_1, S')\rangle \rightsquigarrow (h_1, S_2)}\,Atom}{\langle \texttt{acc(x.f.g)}, (h_0, S_1)\rangle \rightsquigarrow (h_1, S_2)}\,\dagger\,Extract}{\langle \texttt{acc(x.f)} * \texttt{acc(x.f.g)}, (h_0, S_0)\rangle \rightsquigarrow (h_1, S_2)}\,Star$$

The package logic is sound and complete for computing footprints

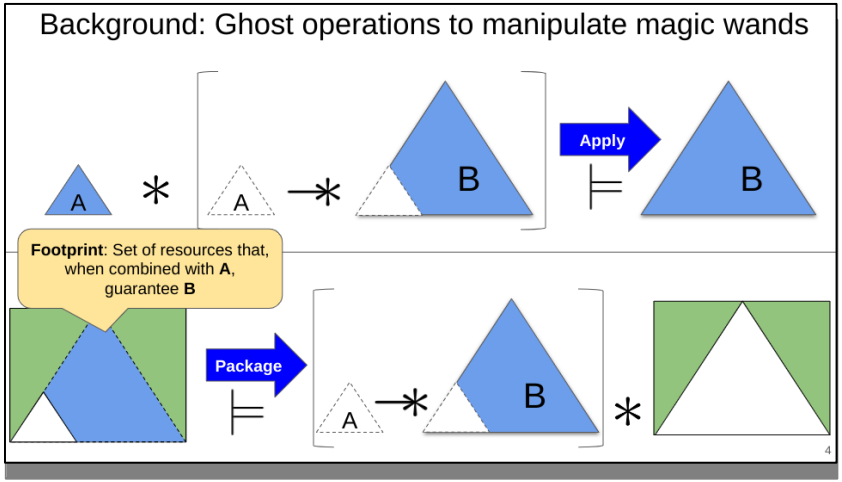


Any valid footprint computation ↔ correct derivation in the package logic

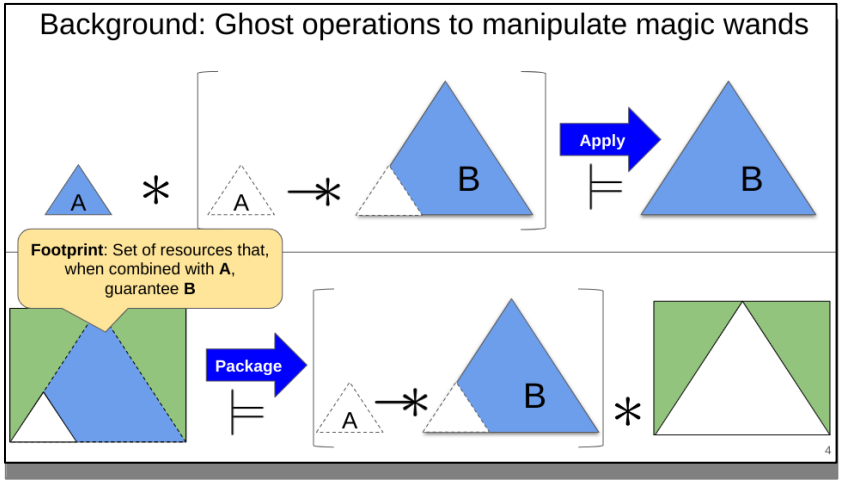
S_0 Set of pairs of states S_1 S_2

Conclusion

Conclusion

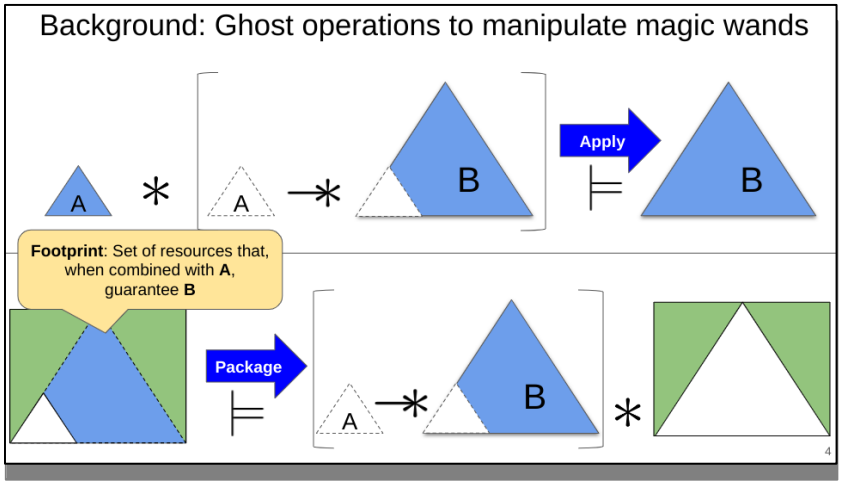

Background: Ghost operations to manipulate magic wands
A
A
B
Apply
B
Footprint: Set of resources that, when combined with A, guarantee B
Package
A
B
4

Conclusion

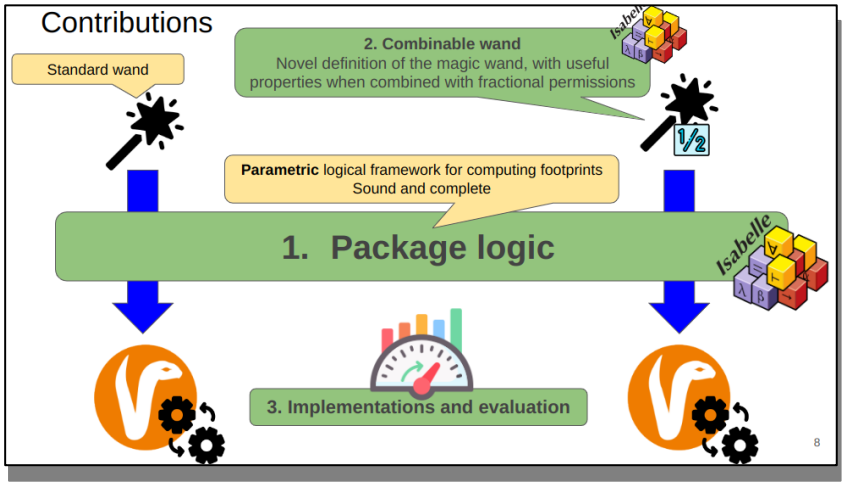

Background: Ghost operations to manipulate magic wands
A
A
B
Apply
B
Footprint: Set of resources that, when combined with A, guarantee B
Package
A
B
4


Challenge: Automatically compute a small, valid footprint
Not automated: User chooses footprint
2015
Witnessing the elimination of magic wands
Stefan Blom · Marieke Huisman
VerCors
Automated
Computed footprint sometimes invalid
2015
Lightweight Support for Magic Wands in an Automatic Verifier
Malte Schwerhoff[1] and Alexander J. Summers[2]
VIPER
6

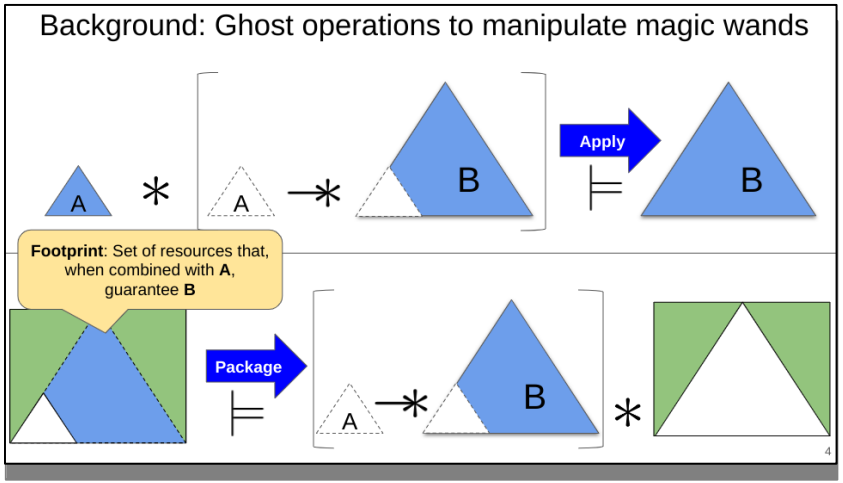
Conclusion


Background: Ghost operations to manipulate magic wands
A
A
B
Apply
B
Footprint: Set of resources that, when combined with A, guarantee B
Package
A
B
4

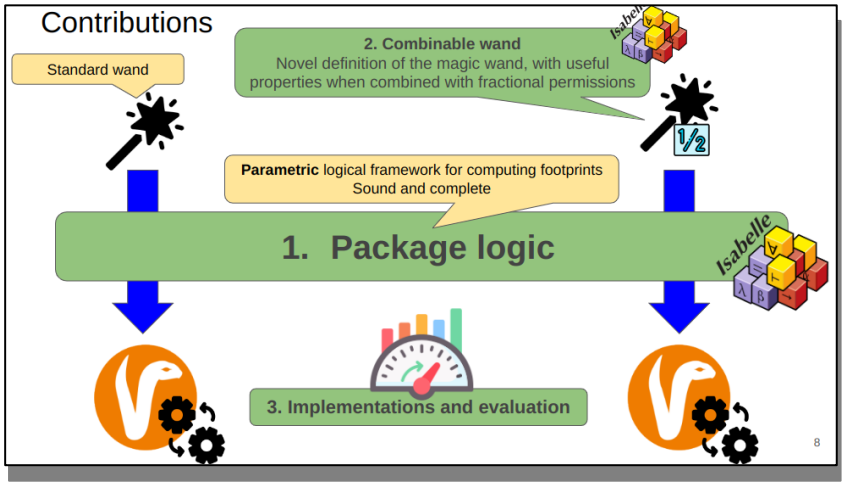

Challenge: Automatically compute a small, valid footprint
Not automated: User chooses footprint
2015
Witnessing the elimination of magic wands
Stefan Blom · Marieke Huisman
VerCors
Automated
Computed footprint sometimes invalid
2015
Lightweight Support for Magic Wands in an Automatic Verifier
Malte Schwerhoff[1] and Alexander J. Summers[2]
VIPER
6


Contributions
Standard wand
2. Combinable wand
Novel definition of the magic wand, with useful properties when combined with fractional permissions
1/2
Parametric logical framework for computing footprints
Sound and complete
1. Package logic
Isabelle
3. Implementations and evaluation
8

Conclusion


Background: Ghost operations to manipulate magic wands
A
A
B
Apply
B
Footprint: Set of resources that, when combined with A, guarantee B
Package
A
B
4


Challenge: Automatically compute a small, valid footprint
Not automated: User chooses footprint
2015
Witnessing the elimination of magic wands
Stefan Blom · Marieke Huisman
VerCors
Automated
Computed footprint sometimes invalid
2015
Lightweight Support for Magic Wands in an Automatic Verifier
Malte Schwerhoff[1] and Alexander J. Summers[2]
VIPER
6


Contributions
Standard wand
2. Combinable wand
Novel definition of the magic wand, with useful properties when combined with fractional permissions
Isabelle
1/2
Parametric logical framework for computing footprints
Sound and complete
1. Package logic
Isabelle
3. Implementations and evaluation
8


CAV
Artifact
Evaluation
Available


CAV
Artifact
Evaluation
Reusable