Observer Symbols

Mattias Ulbrich & Benjamin Weiß

May 18, 2009

Evolution

Location dependent symbols (Richard)

Evolution

Location dependent symbols (Richard)	Verification of modifies clauses (Andreas)

Evolution

Location dependent symbols (Richard)

Verification of modifies clauses (Andreas)

Semantics of anonymising updates (PHS)

Evolution

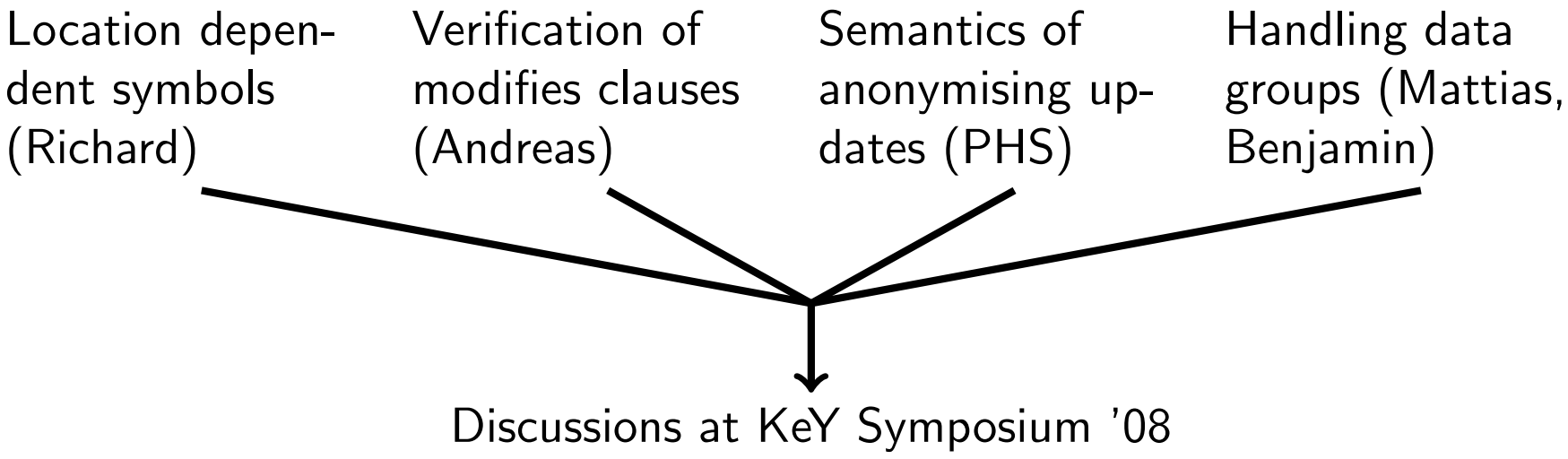
Location dependent symbols (Richard)

Verification of modifies clauses (Andreas)

Semantics of anonymising updates (PHS)

Handling data groups (Mattias, Benjamin)

Evolution


Location dependent symbols (Richard)
Verification of modifies clauses (Andreas)
Semantics of anonymising updates (PHS)
Handling data groups (Mattias, Benjamin)
Discussions at KeY Symposium '08

Evolution

Location dependent symbols (Richard)

Verification of modifies clauses (Andreas)

Semantics of anonymising updates (PHS)

Handling data groups (Mattias, Benjamin)

↓

Discussions at KeY Symposium '08

↓

"Classification of Symbols" (this talk)

Evolution

Location dependent symbols (Richard)

Verification of modifies clauses (Andreas)

Semantics of anonymising updates (PHS)

Handling data groups (Mattias, Benjamin)

↓

Discussions at KeY Symposium '08

↓

"Classification of Symbols" (this talk)

+ Roman's thesis (next talk)

Evolution

Location dependent symbols (Richard)

Verification of modifies clauses (Andreas)

Semantics of anonymising updates (PHS)

Handling data groups (Mattias, Benjamin)

↓

Discussions at KeY Symposium '08

↓

"Classification of Symbols" (this talk)

+ Roman's thesis (next talk)

⇣

Explicit heap model (later talk)

What is this all about?

Reasoning with sets of memory locations:

What is this all about?

Reasoning with sets of memory locations:

- "A state change affects at most locations in L" (*modifies clauses*, *anonymising updates*)

What is this all about?

Reasoning with sets of memory locations:

- "A state change affects at most locations in L"
  (*modifies clauses*, *anonymising updates*)
- "If locations L are not changed, x also cannot be affected"
  (location *dependencies*)

What is this all about?

Reasoning with sets of memory locations:

- "A state change affects at most locations in L" (*modifies clauses*, *anonymising updates*)
- "If locations L are not changed, x also cannot be affected" (location *dependencies*)
- Handle location sets whose contents are not completely known (*data groups*)

Old Situation

Old classification of function & predicate symbols

- Rigid (e.g. $+$)

Old Situation

Old classification of function & predicate symbols

- Rigid (e.g. $+$)
- Non-rigid

Old Situation

Old classification of function & predicate symbols

- Rigid (e.g. $+$)
- Non-rigid
  - Location (e.g. Java fields, `[]`)

Old Situation

Old classification of function & predicate symbols

- Rigid (e.g. $+$)
- Non-rigid
  - Location (e.g. Java fields, `[]`)
  - Location dependent

Old Situation

Old classification of function & predicate symbols

- Rigid (e.g. $+$)
- Non-rigid
  - Location (e.g. Java fields, `[]`)
  - Location dependent
    - Without explicit dependencies

Old Situation

Old classification of function & predicate symbols

- Rigid (e.g. $+$)
- Non-rigid
  - Location (e.g. Java fields, `[]`)
  - Location dependent
    - Without explicit dependencies
    - With explicit dependencies (e.g. $nonNullArray[for\ i; a[i]]$)

Old Situation

Old classification of function & predicate symbols

- Rigid (e.g. $+$)
- Non-rigid
  - Location (e.g. Java fields, `[]`)
  - Location dependent
    - Without explicit dependencies
    - With explicit dependencies (e.g. $nonNullArray[for\ i; a[i]]$)

The distinction "location vs. location dependent" is significant!

Old Situation

Old classification of function & predicate symbols

- Rigid (e.g. $+$)
- Non-rigid
  - Location (e.g. Java fields, `[]`)
  - Location dependent
    - Without explicit dependencies
    - With explicit dependencies (e.g. $nonNullArray[for\ i; a[i]]$)

The distinction "location vs. location dependent" is significant!

$$(\{f := 3\}g) \doteq g$$

Old Situation

Old classification of function & predicate symbols

- Rigid (e.g. $+$)
- Non-rigid
  - Location (e.g. Java fields, `[]`)
  - Location dependent
    - Without explicit dependencies
    - With explicit dependencies (e.g. $nonNullArray[for\ i; a[i]]$)

The distinction "location vs. location dependent" is significant!

$$(\{f := 3\}g) \doteq g$$

- *valid* if g is location symbol

Old Situation

Old classification of function & predicate symbols

- Rigid (e.g. $+$)
- Non-rigid
  - Location (e.g. Java fields, `[]`)
  - Location dependent
    - Without explicit dependencies
    - With explicit dependencies (e.g. $nonNullArray[for\ i; a[i]]$)

The distinction "location vs. location dependent" is significant!

$$(\{f := 3\}g) \doteq g$$

- *valid* if g is location symbol
- *not valid* if g is location dependent symbol

Old Situation (cont.)

We already can...

- describe location sets with location descriptors *ld* (e.g. `o.f`, *for* i; a[i])

Old Situation (cont.)

We already can...

- describe location sets with location descriptors *ld* (e.g. `o.f`, *for* i; a[i])
- verify that a method modifies at most *ld*

Old Situation (cont.)

We already can...

- describe location sets with location descriptors *ld* (e.g. `o.f`, *for* i; a[i])
- verify that a method modifies at most *ld*
- anonymise *ld*

Old Situation (cont.)

We already can...

- describe location sets with location descriptors *ld* (e.g. `o.f`, *for* i; a[i])
- verify that a method modifies at most *ld*
- anonymise *ld*
- define non-rigid symbols that depend at most on *ld*

Old Situation (cont.)

We already can...

- describe location sets with location descriptors *ld* (e.g. `o.f`, *for* i; a[i])
- verify that a method modifies at most *ld*
- anonymise *ld*
- define non-rigid symbols that depend at most on *ld*

We cannot...

- do any of this without explicitly naming all involved location symbols

Data Abstraction

```
interface List {



  void add(Object element);


  Object get(int index);
}
```

Data Abstraction

```
interface List {


  //@ assignable ?
  void add(Object element);


  Object get(int index);
}
```

Data Abstraction

```
interface List {
  //@ model instance JMLObjectSequence elements;

  //@ assignable ?
  void add(Object element);


  Object get(int index);
}
```

Data Abstraction

```
interface List {
  //@ model instance JMLObjectSequence elements;

  //@ assignable elements;
  void add(Object element);


  Object get(int index);
}
```

Data Abstraction

```
interface List {
  //@ model instance JMLObjectSequence elements;

  //@ assignable elements;
  void add(Object element);


  /*@pure@*/ Object get(int index);
}
```

Data Abstraction

```
interface List {
  //@ model instance JMLObjectSequence elements;

  //@ assignable elements;
  void add(Object element);

  //@ accessible elements;
  /*@pure@*/ Object get(int index);
}
```

Data Abstraction

```
interface List {
  //@ model instance JMLObjectSequence elements;

  //@ assignable elements;
  void add(Object element);

  //@ accessible elements;
  /*@pure@*/ Object get(int index);
}


class ArrayList implements List {



  ...
}
```

Data Abstraction

```
interface List {
  //@ model instance JMLObjectSequence elements;

  //@ assignable elements;
  void add(Object element);

  //@ accessible elements;
  /*@pure@*/ Object get(int index);
}


class ArrayList implements List {
  Object[] arr;

  ...
}
```

Data Abstraction

```
interface List {
  //@ model instance JMLObjectSequence elements;

  //@ assignable elements;
  void add(Object element);

  //@ accessible elements;
  /*@pure@*/ Object get(int index);
}


class ArrayList implements List {
  Object[] arr; //@ in elements;

  ...
}
```

Data Abstraction

```
interface List {
  //@ model instance JMLObjectSequence elements;

  //@ assignable elements;
  void add(Object element);

  //@ accessible elements;
  /*@pure@*/ Object get(int index);
}


class ArrayList implements List {
  Object[] arr; //@ in elements;
                //@ maps arr[*] into elements;
  ...
}
```

Data Abstraction

```
interface List {
  //@ model instance JMLObjectSequence elements;

  //@ assignable elements;
  void add(Object element);

  //@ accessible elements;
  /*@pure@*/ Object get(int index);
}


class ArrayList implements List {
  Object[] arr; //@ in elements;
                //@ maps arr[*] into elements;
  ...
}
```

Syntax

Signature

Contains location symbols and observer symbols

Syntax

Signature

Contains location symbols and observer symbols

Location symbols f...

- "constitute" the state

Syntax

Signature

Contains location symbols and observer symbols

Location symbols f...

- "constitute" the state
- can be updated

Syntax

Signature

Contains location symbols and observer symbols

Location symbols f...

- "constitute" the state
- can be updated
- examples: Java fields, `[]`, JML ghost fields, ...

Syntax

Signature

Contains location symbols and observer symbols

Location symbols f...

- "constitute" the state
- can be updated
- examples: Java fields, `[]`, JML ghost fields, ...

Observer symbols *obs*...

- "observe" the state

Syntax

Signature

Contains location symbols and observer symbols

Location symbols f...

- "constitute" the state
- can be updated
- examples: Java fields, [], JML ghost fields, ...

Observer symbols *obs*...

- "observe" the state
- have dual semantics: a value v and a set of locations L

Syntax

Signature

Contains location symbols and observer symbols

Location symbols f...

- "constitute" the state
- can be updated
- examples: Java fields, `[]`, JML ghost fields, ...

Observer symbols *obs*...

- "observe" the state
- have dual semantics: a value v and a set of locations L
  - both v and L are state dependent

Syntax

Signature

Contains location symbols and observer symbols

Location symbols f...

- "constitute" the state
- can be updated
- examples: Java fields, `[]`, JML ghost fields, ...

Observer symbols *obs*...

- "observe" the state
- have dual semantics: a value v and a set of locations L
  - both v and L are state dependent
  - if the values in L do not change, then v does not change

Syntax

Signature

Contains location symbols and observer symbols

Location symbols f...

- "constitute" the state
- can be updated
- examples: Java fields, [], JML ghost fields, ...

Observer symbols *obs*...

- "observe" the state
- have dual semantics: a value v and a set of locations L
  - both v and L are state dependent
  - if the values in L do not change, then v does not change
  - can be made "rigid" / "blind" by postulating $L = \emptyset$

Syntax

Signature

Contains location symbols and observer symbols

Location symbols f...

- "constitute" the state
- can be updated
- examples: Java fields, [], JML ghost fields, ...

Observer symbols *obs*...

- "observe" the state
- have dual semantics: a value v and a set of locations L
  - both v and L are state dependent
  - if the values in L do not change, then v does not change
  - can be made "rigid" / "blind" by postulating $L = \emptyset$
- examples: $+$, *reach*, *inReachableState*, queries, JML model fields, ...

Syntax (cont.)

Location descriptors

- *empty*, *everything*

Syntax (cont.)

Location descriptors

- *empty*, *everything*
- $f(\bar{t})$

Syntax (cont.)

Location descriptors

- *empty*, *everything*
- $f(\bar{t})$
- $obs(\bar{t})$

Syntax (cont.)

Location descriptors

- *empty*, *everything*
- $f(\bar{t})$
- $obs(\bar{t})$
- $(ld_1 \parallel ld_2)$, $(if\,\varphi; ld)$, $(for\,x; ld)$

Syntax (cont.)

Location descriptors

- *empty*, *everything*
- $f(\bar{t})$
- $obs(\bar{t})$
- $(ld_1 \parallel ld_2)$, $(if\,\varphi; ld)$, $(for\,x; ld)$

Terms, formulas

$locSubset(ld_1, ld_2)$,

Syntax (cont.)

Location descriptors

- $empty$, $everything$
- $f(\bar{t})$
- $obs(\bar{t})$
- $(ld_1 \parallel ld_2)$, $(if\,\varphi; ld)$, $(for\,x; ld)$

Terms, formulas

$locSubset(ld_1, ld_2)$, $locEqual(ld_1, ld_2)$,

Syntax (cont.)

Location descriptors

- *empty*, *everything*
- $f(\bar{t})$
- $obs(\bar{t})$
- $(ld_1 \parallel ld_2)$, $(if\,\varphi; ld)$, $(for\,x; ld)$

Terms, formulas

$locSubset(ld_1, ld_2)$, $locEqual(ld_1, ld_2)$, $locDisjoint(ld_1, ld_2)$

Syntax (cont.)

Location descriptors

- *empty*, *everything*
- $f(\bar{t})$
- $obs(\bar{t})$
- $(ld_1 \parallel ld_2)$, $(if\,\varphi; ld)$, $(for\,x; ld)$

Terms, formulas

$locSubset(ld_1, ld_2)$, $locEqual(ld_1, ld_2)$, $locDisjoint(ld_1, ld_2)$

Updates

$ld := *n$ for every $n \in \mathbb{N}$

Semantics

Locations

$(f, \bar{v})$	where f location symbol, $\bar{v}$ values

Semantics

Locations

$(f, \bar{v})$ where f location symbol, $\bar{v}$ values

Kripke structures

$(\mathcal{D}, \mathcal{S}, \rho, *, \mathit{observe}, \mathit{depends})$

Semantics

Locations

$(f, \bar{v})$ where f location symbol, $\bar{v}$ values

Kripke structures

$(\mathcal{D}, \mathcal{S}, \rho, *, \mathit{observe}, \mathit{depends})$

- $\mathcal{D}$: universe

Semantics

Locations

$(f, \bar{v})$ where f location symbol, $\bar{v}$ values

Kripke structures

$(\mathcal{D}, \mathcal{S}, \rho, *, \mathit{observe}, \mathit{depends})$

- $\mathcal{D}$: universe
- $\mathcal{S}$: all interpretations of the location symbols ("states")

Semantics

Locations

$(f, \bar{v})$ where f location symbol, $\bar{v}$ values

Kripke structures

$(\mathcal{D}, \mathcal{S}, \rho, *, \mathit{observe}, \mathit{depends})$

- $\mathcal{D}$: universe
- $\mathcal{S}$: all interpretations of the location symbols ("states")
- $\rho : \mathit{Programs} \rightarrow \mathcal{S}^2$

Semantics

Locations

$(f, \bar{v})$ where f location symbol, $\bar{v}$ values

Kripke structures

$(\mathcal{D}, \mathcal{S}, \rho, *, \mathit{observe}, \mathit{depends})$

- $\mathcal{D}$: universe
- $\mathcal{S}$: all interpretations of the location symbols ("states")
- $\rho : \mathit{Programs} \to \mathcal{S}^2$
- $* : \mathbb{N} \to \mathcal{S}$

Semantics

Locations

$(f, \bar{v})$ where f location symbol, $\bar{v}$ values

Kripke structures

$(\mathcal{D}, \mathcal{S}, \rho, *, \mathit{observe}, \mathit{depends})$

- $\mathcal{D}$: universe
- $\mathcal{S}$: all interpretations of the location symbols ("states")
- $\rho : \mathit{Programs} \rightarrow \mathcal{S}^2$
- $* : \mathbb{N} \rightarrow \mathcal{S}$
- $\mathit{observe}(s, \mathit{obs}) : \mathcal{D}^n \rightarrow \mathcal{D}$

Semantics

Locations

$(f, \bar{v})$ where f location symbol, $\bar{v}$ values

Kripke structures

$(\mathcal{D}, \mathcal{S}, \rho, *, \mathit{observe}, \mathit{depends})$

- $\mathcal{D}$: universe
- $\mathcal{S}$: all interpretations of the location symbols ("states")
- $\rho : \mathit{Programs} \rightarrow \mathcal{S}^2$
- $* : \mathbb{N} \rightarrow \mathcal{S}$
- $\mathit{observe}(s, \mathit{obs}) : \mathcal{D}^n \rightarrow \mathcal{D}$
- $\mathit{depends}(s, \mathit{obs}) : \mathcal{D}^n \rightarrow 2^{\mathit{Locations}}$

Semantics

Locations

$(f, \bar{v})$ where f location symbol, $\bar{v}$ values

Kripke structures

$(\mathcal{D}, \mathcal{S}, \rho, *, observe, depends)$

- $\mathcal{D}$: universe
- $\mathcal{S}$: all interpretations of the location symbols ("states")
- $\rho : Programs \rightarrow \mathcal{S}^2$
- $* : \mathbb{N} \rightarrow \mathcal{S}$
- $observe(s, obs) : \mathcal{D}^n \rightarrow \mathcal{D}$
- $depends(s, obs) : \mathcal{D}^n \rightarrow 2^{Locations}$

If for all $l \in depends(s, obs)(\bar{v})$: $s(l) = s'(l)$

Semantics

Locations

$(f, \bar{v})$ where f location symbol, $\bar{v}$ values

Kripke structures

$(\mathcal{D}, \mathcal{S}, \rho, *, observe, depends)$

- $\mathcal{D}$: universe
- $\mathcal{S}$: all interpretations of the location symbols ("states")
- $\rho : Programs \to \mathcal{S}^2$
- $* : \mathbb{N} \to \mathcal{S}$
- $observe(s, obs) : \mathcal{D}^n \to \mathcal{D}$
- $depends(s, obs) : \mathcal{D}^n \to 2^{Locations}$

If for all $l \in depends(s, obs)(\bar{v})$: $s(l) = s'(l)$
then $observe(s, obs)(\bar{v}) = observe(s', obs)(\bar{v})$

Semantics (cont.)

Location descriptors

- $val_{\mathcal{K},s}(empty) = \emptyset$

Semantics (cont.)

Location descriptors

- $val_{\mathcal{K},s}(empty) = \emptyset$
- $val_{\mathcal{K},s}(everything) = Locations$

Semantics (cont.)

Location descriptors

- $val_{\mathcal{K},s}(empty) = \emptyset$
- $val_{\mathcal{K},s}(everything) = Locations$
- $val_{\mathcal{K},s}(f(\bar{t})) = \{(f, val_{\mathcal{K},s}(\bar{t}))\}$

Semantics (cont.)

Location descriptors

- $val_{\mathcal{K},s}(empty) = \emptyset$
- $val_{\mathcal{K},s}(everything) = Locations$
- $val_{\mathcal{K},s}(f(\bar{t})) = \{(f, val_{\mathcal{K},s}(\bar{t}))\}$
- $val_{\mathcal{K},s}(obs(\bar{t})) = depends(s, obs)(val_{\mathcal{K},s}(\bar{t}))$

Semantics (cont.)

Location descriptors

- $val_{\mathcal{K},s}(empty) = \emptyset$
- $val_{\mathcal{K},s}(everything) = Locations$
- $val_{\mathcal{K},s}(f(\bar{t})) = \{(f, val_{\mathcal{K},s}(\bar{t}))\}$
- $val_{\mathcal{K},s}(obs(\bar{t})) = depends(s, obs)(val_{\mathcal{K},s}(\bar{t}))$
- ...

Semantics (cont.)

Location descriptors

- $val_{\mathcal{K},s}(empty) = \emptyset$
- $val_{\mathcal{K},s}(everything) = Locations$
- $val_{\mathcal{K},s}(f(\bar{t})) = \{(f, val_{\mathcal{K},s}(\bar{t}))\}$
- $val_{\mathcal{K},s}(obs(\bar{t})) = depends(s, obs)(val_{\mathcal{K},s}(\bar{t}))$
- ...

Terms, formulas

- $\mathcal{K}, s \models locSubset(ld_1, ld_2) \quad \text{iff} \quad val_{\mathcal{K},s}(ld_1) \subseteq val_{\mathcal{K},s}(ld_2)$

Semantics (cont.)

Location descriptors

- $val_{\mathcal{K},s}(empty) = \emptyset$
- $val_{\mathcal{K},s}(everything) = Locations$
- $val_{\mathcal{K},s}(f(\bar{t})) = \{(f, val_{\mathcal{K},s}(\bar{t}))\}$
- $val_{\mathcal{K},s}(obs(\bar{t})) = depends(s, obs)(val_{\mathcal{K},s}(\bar{t}))$
- ...

Terms, formulas

- $\mathcal{K}, s \models locSubset(ld_1, ld_2) \quad \text{iff} \quad val_{\mathcal{K},s}(ld_1) \subseteq val_{\mathcal{K},s}(ld_2)$
- ...

Semantics (cont.)

Location descriptors

- $val_{\mathcal{K},s}(empty) = \emptyset$
- $val_{\mathcal{K},s}(everything) = Locations$
- $val_{\mathcal{K},s}(f(\bar{t})) = \{(f, val_{\mathcal{K},s}(\bar{t}))\}$
- $val_{\mathcal{K},s}(obs(\bar{t})) = depends(s, obs)(val_{\mathcal{K},s}(\bar{t}))$
- ...

Terms, formulas

- $\mathcal{K}, s \models locSubset(ld_1, ld_2)$ iff $val_{\mathcal{K},s}(ld_1) \subseteq val_{\mathcal{K},s}(ld_2)$
- ...

Updates

$ld := *n$ sets all locations in $val_{\mathcal{K},s}(ld)$ to state $*(n)$

Examples

Formula	*Valid?*
$(\{f := 3\}g) \doteq g$	

Examples

Formula	*Valid?*
$(\{f := 3\}g) \doteq g$	✓

Examples

Formula	*Valid?*
$(\{f := 3\}g) \doteq g$	✓
$(\{f := 3\}obs) \doteq obs$	

Examples

Formula	*Valid?*
$(\{f := 3\}g) \doteq g$	✓
$(\{f := 3\}obs) \doteq obs$	✗

Examples

Formula	*Valid?*
$(\{f := 3\}g) \doteq g$	✓
$(\{f := 3\}obs) \doteq obs$	✗
$locEqual(obs, empty) \rightarrow (\{f := 3\}obs) \doteq obs$	

Examples

Formula	*Valid?*
$(\{f := 3\}g) \doteq g$	✓
$(\{f := 3\}obs) \doteq obs$	✗
$locEqual(obs, empty) \rightarrow (\{f := 3\}obs) \doteq obs$	✓

Examples

Formula	*Valid?*
$(\{f := 3\}g) \doteq g$	✓
$(\{f := 3\}obs) \doteq obs$	✗
$locEqual(obs, empty) \rightarrow (\{f := 3\}obs) \doteq obs$	✓
$locDisjoint(obs, f) \rightarrow (\{f := 3\}obs) \doteq obs$	

Examples

Formula	*Valid?*
$(\{f := 3\}g) \doteq g$	✓
$(\{f := 3\}obs) \doteq obs$	✗
$locEqual(obs, empty) \rightarrow (\{f := 3\}obs) \doteq obs$	✓
$locDisjoint(obs, f) \rightarrow (\{f := 3\}obs) \doteq obs$	✓

Examples

Formula	*Valid?*
$(\{f := 3\}g) \doteq g$	✓
$(\{f := 3\}obs) \doteq obs$	✗
$locEqual(obs, empty) \rightarrow (\{f := 3\}obs) \doteq obs$	✓
$locDisjoint(obs, f) \rightarrow (\{f := 3\}obs) \doteq obs$	✓
$(\{obs := *1\}f) \doteq f$	

Examples

Formula	*Valid?*
$(\{f := 3\}g) \doteq g$	✓
$(\{f := 3\}obs) \doteq obs$	✗
$locEqual(obs, empty) \rightarrow (\{f := 3\}obs) \doteq obs$	✓
$locDisjoint(obs, f) \rightarrow (\{f := 3\}obs) \doteq obs$	✓
$(\{obs := *1\}f) \doteq f$	✗

Examples

Formula	*Valid?*
$(\{f := 3\}g) \doteq g$	✓
$(\{f := 3\}obs) \doteq obs$	✗
$locEqual(obs, empty) \rightarrow (\{f := 3\}obs) \doteq obs$	✓
$locDisjoint(obs, f) \rightarrow (\{f := 3\}obs) \doteq obs$	✓
$(\{obs := *1\}f) \doteq f$	✗
$locDisjoint(obs, f) \rightarrow (\{obs := *1\}f) \doteq f$	

Examples

Formula	*Valid?*
$(\{f := 3\}g) \doteq g$	✓
$(\{f := 3\}obs) \doteq obs$	✗
$locEqual(obs, empty) \rightarrow (\{f := 3\}obs) \doteq obs$	✓
$locDisjoint(obs, f) \rightarrow (\{f := 3\}obs) \doteq obs$	✓
$(\{obs := *1\}f) \doteq f$	✗
$locDisjoint(obs, f) \rightarrow (\{obs := *1\}f) \doteq f$	✓

Summary

- New symbol classification: location vs. observer

Summary

- New symbol classification: location vs. observer
- New syntactical constructs:
  - observer symbols in location descriptors

Summary

- New symbol classification: location vs. observer
- New syntactical constructs:
  - observer symbols in location descriptors
  - talking about location descriptors in formulas

Summary

- New symbol classification: location vs. observer
- New syntactical constructs:
  - observer symbols in location descriptors
  - talking about location descriptors in formulas
  - generic anonymising updates $ld := *n$

Summary

- New symbol classification: location vs. observer
- New syntactical constructs:
  - observer symbols in location descriptors
  - talking about location descriptors in formulas
  - generic anonymising updates $ld := *n$
- Allows to reason with only partially specified location sets ($\rightsquigarrow$ modularity)

Summary

- New symbol classification: location vs. observer
- New syntactical constructs:
  - observer symbols in location descriptors
  - talking about location descriptors in formulas
  - generic anonymising updates $ld := *n$
- Allows to reason with only partially specified location sets ($\rightsquigarrow$ modularity)
- Related work: data groups, ownership, regional logic, separation logic, *dynamic frames*

Summary

- New symbol classification: location vs. observer
- New syntactical constructs:
  - observer symbols in location descriptors
  - talking about location descriptors in formulas
  - generic anonymising updates $ld := *n$
- Allows to reason with only partially specified location sets ($\rightsquigarrow$ modularity)
- Related work: data groups, ownership, regional logic, separation logic, *dynamic frames*
- Expressive enough?

Summary

- New symbol classification: location vs. observer
- New syntactical constructs:
  - observer symbols in location descriptors
  - talking about location descriptors in formulas
  - generic anonymising updates $ld := *n$
- Allows to reason with only partially specified location sets ($\rightsquigarrow$ modularity)
- Related work: data groups, ownership, regional logic, separation logic, *dynamic frames*
- Expressive enough?
  - cannot give dependencies of location sets themselves

Summary

- New symbol classification: location vs. observer
- New syntactical constructs:
  - observer symbols in location descriptors
  - talking about location descriptors in formulas
  - generic anonymising updates $ld := *n$
- Allows to reason with only partially specified location sets ($\rightsquigarrow$ modularity)
- Related work: data groups, ownership, regional logic, separation logic, *dynamic frames*
- Expressive enough?
  - cannot give dependencies of location sets themselves
  - cannot say how a location set may be changed by a program

Summary

- New symbol classification: location vs. observer
- New syntactical constructs:
  - observer symbols in location descriptors
  - talking about location descriptors in formulas
  - generic anonymising updates $ld := *n$
- Allows to reason with only partially specified location sets ($\rightsquigarrow$ modularity)
- Related work: data groups, ownership, regional logic, separation logic, *dynamic frames*
- Expressive enough?
  - cannot give dependencies of location sets themselves
  - cannot say how a location set may be changed by a program
  - cannot quantify over locations

Summary

- New symbol classification: location vs. observer
- New syntactical constructs:
  - observer symbols in location descriptors
  - talking about location descriptors in formulas
  - generic anonymising updates $ld := *n$
- Allows to reason with only partially specified location sets ($\rightsquigarrow$ modularity)
- Related work: data groups, ownership, regional logic, separation logic, *dynamic frames*
- Expressive enough?
  - cannot give dependencies of location sets themselves
  - cannot say how a location set may be changed by a program
  - cannot quantify over locations
- Calculus?

Summary

- New symbol classification: location vs. observer
- New syntactical constructs:
  - observer symbols in location descriptors
  - talking about location descriptors in formulas
  - generic anonymising updates $ld := *n$
- Allows to reason with only partially specified location sets ($\rightsquigarrow$ modularity)
- Related work: data groups, ownership, regional logic, separation logic, *dynamic frames*
- Expressive enough?
  - cannot give dependencies of location sets themselves
  - cannot say how a location set may be changed by a program
  - cannot quantify over locations
- Calculus? $\rightsquigarrow$ next talk!