Re-classification and Multi-threading: $\mathcal{F}ickle_{\mathtt{MT}}$

MYTHS/MIKADO/DART Meeting, Venice, June 14, 2004

joint work with Ferruccio Damiani and Paola Giannini

Contents of the talk

- Introduction and Motivation through examples
- Operational Semantics
- Type System
- Soundness
- Translation

Example

Frogs and princes play. Frogs blow up their pouch when woken up, princes swing their sword.

Example

Frogs and princes play. Frogs blow up their pouch when woken up, princes swing their sword.

class Player{ **int** age; **void** wake(){...} }

Example

Frogs and princes play. Frogs blow up their pouch when woken up, princes swing their sword.

```
class Player{ int age;   void wake(){...}   }

class Frog extends Player{                class Prince extends Player{
  Vocal pouch;                              Weapon sword;
  void wake(){pouch.blow() }                void wake(){sword.swing()}
}                                         }
```

Example

Frogs and princes play. Frogs blow up their pouch when woken up, princes swing their sword.

```
class  Player{ int age;    void wake(){...}   }
```

```
class Frog extends Player{
    Vocal pouch;
    void wake(){pouch.blow() }
}
```

```
class Prince extends Player{
    Weapon sword;
    void wake(){sword.swing()}
}
```

but then, ...

Example cont.

Frogs turn into princes when kissed

Example cont.

Frogs turn into princes when kissed

root class $\mathsf{Player}\{\mathbf{int}\ \mathsf{age};\quad \mathbf{void}\ \mathsf{wake}()\{\ldots\}\ \ \mathbf{void}\ \mathsf{kissed}()\{\ldots\}\ \ \}$

Example cont.

Frogs turn into princes when kissed

```
root class  Player{int age;  void wake(){...}  void kissed(){...}  }

state class Frog extends Player{
    Vocal pouch;
    void wake(){pouch.blow() }
    void kissed(){this⇓Prince}
}

state class Prince extends Player{
    Weapon sword;
    void wake(){sword.swing()}
    void kissed(){...}
}
```

Example cont.

Frogs turn into princes when kissed

```
root class  Player{int age;  void wake(){...}  void kissed(){...}  }
```

```
state class Frog extends Player{
    Vocal pouch;
    void wake(){pouch.blow() }
    void kissed(){this⇓Prince}
}
```

```
state class Prince extends Player{
    Weapon sword;
    void wake(){sword.swing()}
    void kissed(){...}
}
```

```
Player p1, p2;  p1:= new Frog;  p2:= p1 ;
```

Example cont.

Frogs turn into princes when kissed

```
root class  Player{int age;  void wake(){...}  void kissed(){...}  }

state class Frog extends Player{
    Vocal pouch;
    void wake(){pouch.blow() }
    void kissed(){this⇓Prince}
}

state class Prince extends Player{
    Weapon sword;
    void wake(){sword.swing()}
    void kissed(){...}
}

Player p1, p2;  p1:= new Frog;  p2:= p1 ;
p2.wake();    blow up pouch
```

Example cont.

Frogs turn into princes when kissed

```
root class  Player{int age;  void wake(){...}  void kissed(){...}  }
```

```
state class Frog extends Player{
    Vocal pouch;
    void wake(){pouch.blow() }
    void kissed(){this⇓Prince}
}
```

```
state class Prince extends Player{
    Weapon sword;
    void wake(){sword.swing()}
    void kissed(){...}
}
```

```
Player p1, p2;  p1:= new Frog;  p2:= p1 ;
p2.wake();     blow up pouch
p1.kissed();      p1 and p2  turn into prince
```

Example cont.

Frogs turn into princes when kissed

```
root class  Player{int age;  void wake(){...}  void kissed(){...}  }

state class Frog extends Player{
    Vocal pouch;
    void wake(){pouch.blow() }
    void kissed(){this⇓Prince}
}

state class Prince extends Player{
    Weapon sword;
    void wake(){sword.swing()}
    void kissed(){...}
}

Player p1, p2;  p1:= new Frog;  p2:= p1 ;
p2.wake();     blow up pouch
p1.kissed();     p1 and p2  turn into prince
p2.wake();     swing sword
```

Reclassification

$\mathcal{F}ickle$ = minimal Java-like language extended with

Reclassification

$\mathcal{F}ickle$ = minimal Java-like language extended with

- operation `this`$\Downarrow$c sets class of `this` to c (p1.kiss() reclassifies p1 and all its aliases to Prince);

Reclassification

$\mathcal{F}ickle$ = minimal Java-like language extended with

- operation `this`$\Downarrow$c sets class of `this` to c (p1.kiss() reclassifies p1 and all its aliases to Prince);
- state-classes (Frog, Prince), whose objects may be re-classified;

Reclassification

$\mathcal{F}ickle$ = minimal Java-like language extended with

- operation `this`$\Downarrow$c sets class of `this` to c (p1.kiss() reclassifies p1 and all its aliases to Prince);
- state-classes (Frog, Prince), whose objects may be re-classified;
- root-classes (Player), the superclasses of state-classes;

Reclassification

$\mathcal{F}ickle$ = minimal Java-like language extended with

- operation `this`$\Downarrow$c sets class of `this` to c (p1.kiss() reclassifies p1 and all its aliases to Prince);
- state-classes (Frog, Prince), whose objects may be re-classified;
- root-classes (Player), the superclasses of state-classes;
- reclassification restricted across subclasses of same root-class.

Reclassification

$\mathcal{F}ickle$ = minimal Java-like language extended with

- operation `this`⇓c sets class of `this` to c (p1.kiss() reclassifies p1 and all its aliases to Prince);
- state-classes (Frog, Prince), whose objects may be re-classified;
- root-classes (Player), the superclasses of state-classes;
- reclassification restricted across subclasses of same root-class.

This allows us to express frogs turning into princes, windows being iconified and expanded, empty stacks becoming non-empty and non-empty stacks becoming empty, *etc.*, *etc.*, *etc.*

Multi-threading: $\mathcal{F}ickle_{\mathtt{MT}}$

spawn(e) starts the evaluation of the expression e in a new thread

Multi-threading: $\mathcal{F}ickle_{\texttt{MT}}$

spawn(e) starts the evaluation of the expression e in a new thread

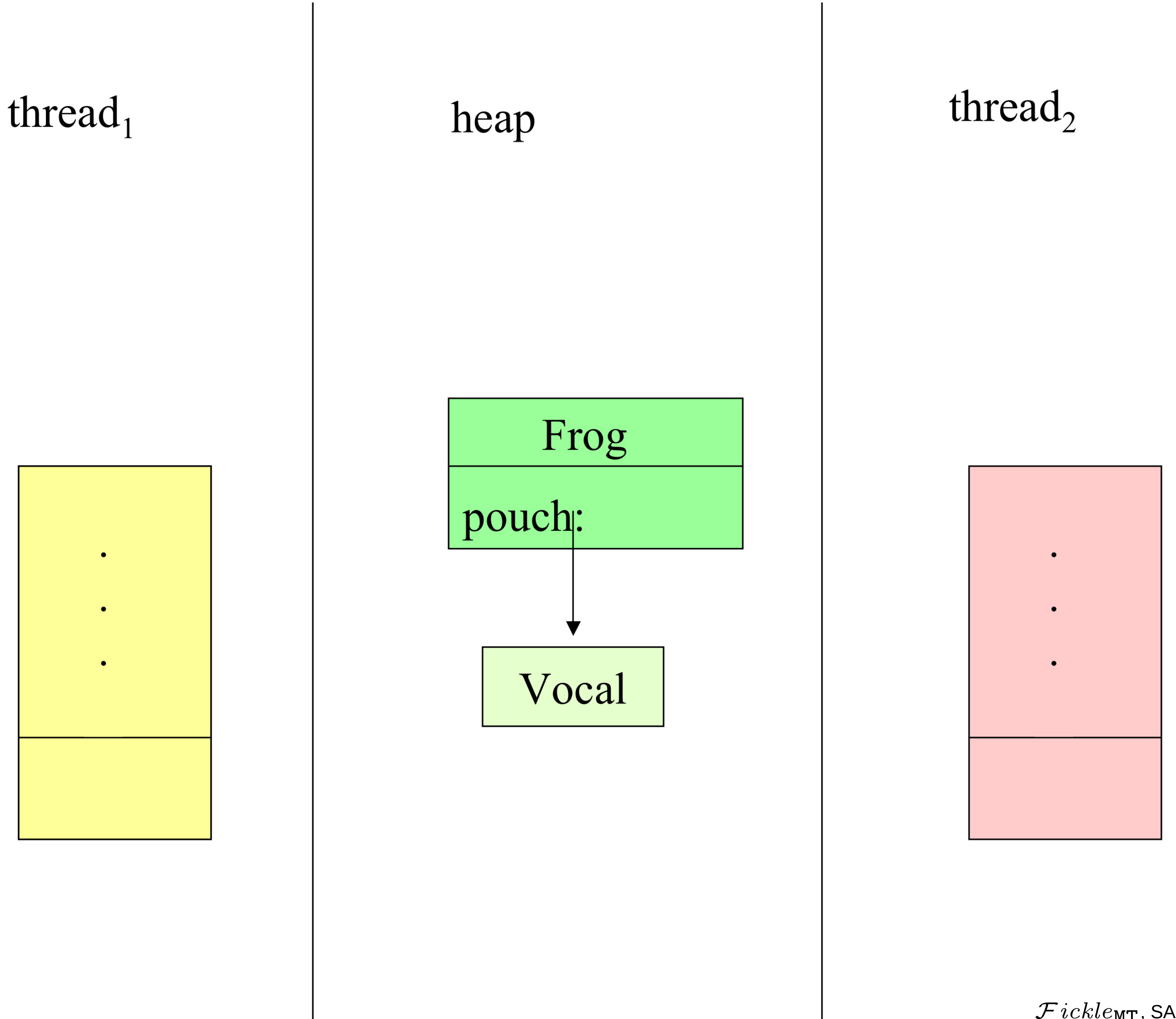
```
class Game extends Object{
    bool play(Player x)
        {spawn(x.wake());
         spawn(x.kissed())}
}
```

Multi-threading: $\mathcal{F}ickle_{\texttt{MT}}$

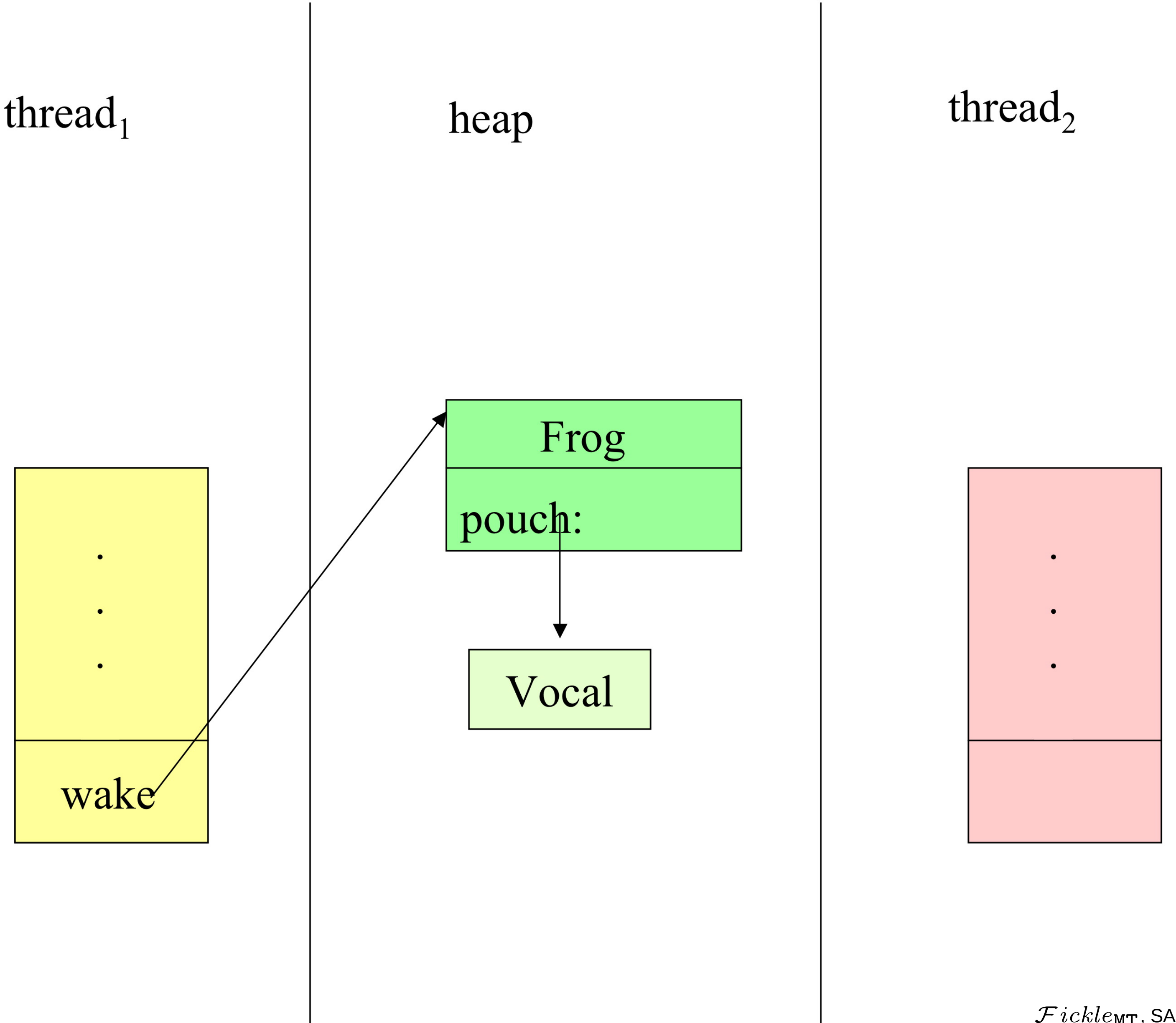
spawn(e) starts the evaluation of the expression e in a new thread

```
class Game extends Object{
    bool play(Player x)
        {spawn(x.wake());
         spawn(x.kissed())}
}
          (new Game).play(new Frog)
```

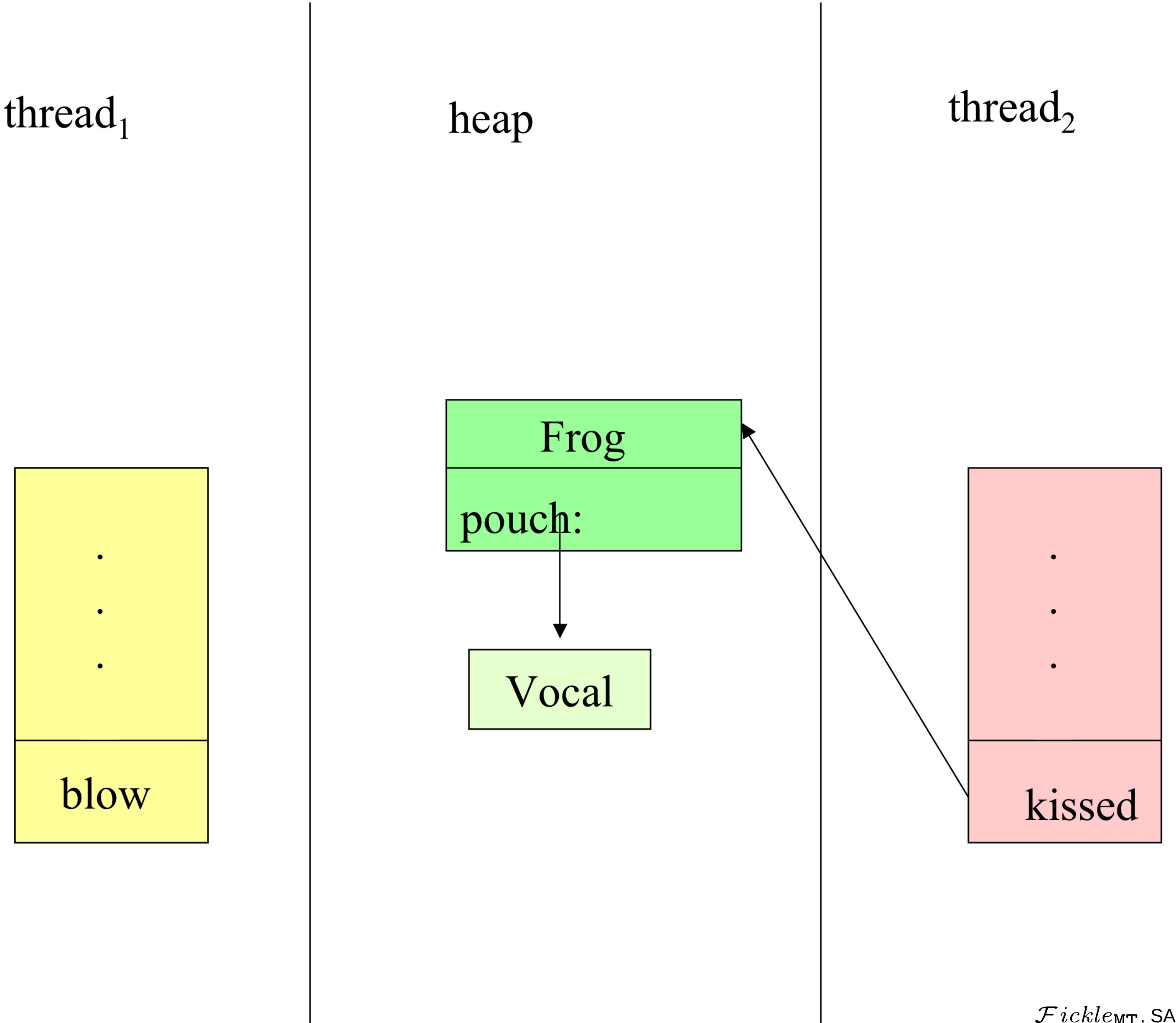
Key Example



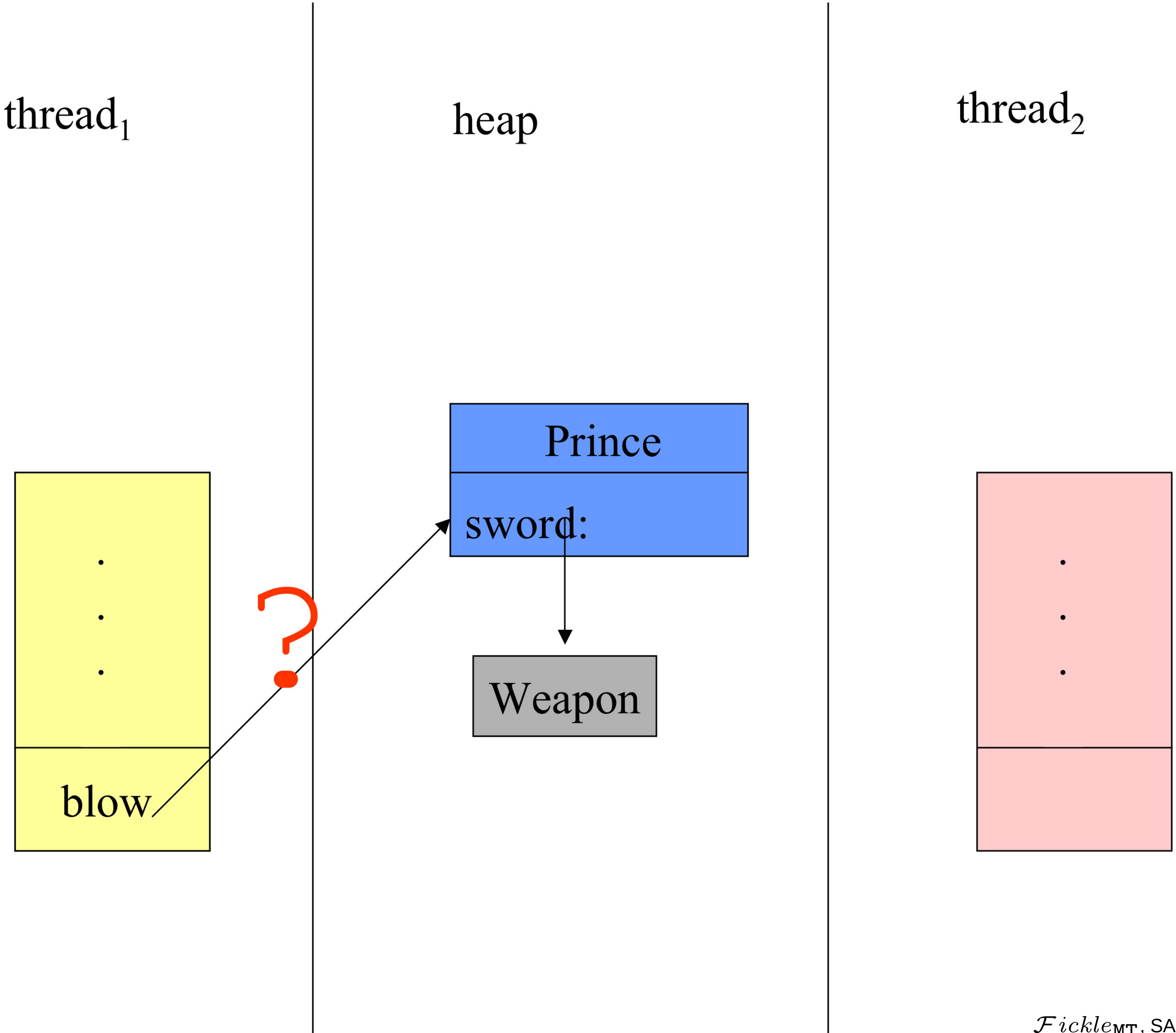
Key Example



Key Example



Key Example



The Problem

Prevent executions in which one thread reclassifies an object while another one is executing a method on the same object

The Problem

Prevent executions in which one thread reclassifies an object while another one is executing a method on the same object

without causing deadlocks

Our Solution

- We need to know:
    - the objects that may be reclassified by a thread
    - the re-classifiable objects that are used as receivers of method calls by a thread

Our Solution

- We need to know:
  - the objects that may be reclassified by a thread
  - the re-classifiable objects that are used as receivers of method calls by a thread
- The type system gathers information on the type of objects reclassified and/or used as receivers

Our Solution

- We need to know:
  - the objects that may be reclassified by a thread
  - the re-classifiable objects that are used as receivers of method calls by a thread
- The type system gathers information on the type of objects reclassified and/or used as receivers
- The operational semantics uses this static information to block threads that could cause errors (dynamic check)

Our Solution

- We need to know:
  - the objects that may be reclassified by a thread
  - the re-classifiable objects that are used as receivers of method calls by a thread
- The type system gathers information on the type of objects reclassified and/or used as receivers
- The operational semantics uses this static information to block threads that could cause errors (dynamic check)
- This blocking threads does not cause deadlock

Our Solution

- We need to know:
  - the objects that may be reclassified by a thread
  - the re-classifiable objects that are used as receivers of method calls by a thread
- The type system gathers information on the type of objects reclassified and/or used as receivers
- The operational semantics uses this static information to block threads that could cause errors (dynamic check)
- This blocking threads does not cause deadlock : a top-level method call acquires the right to have all the objects that may be reclassified or be method call's receivers during its call.

Contents of the talk

- Introduction and Motivation through examples
- Operational Semantics
- Type System
- Soundness
- Translation

Method Declaration

$$\mathsf{t}\ \mathsf{m}\ (\mathsf{t}'\ \mathsf{x})\ \Theta\ \ \{\ \mathsf{e}\ \}$$

Method Declaration

$$\mathsf{t}\ \mathsf{m}\ (\mathsf{t}'\ \mathsf{x})\ \Theta\ \{\ \mathsf{e}\ \}$$

- t is the result type

Method Declaration

$$\mathsf{t}\ \mathsf{m}\ (\mathsf{t}'\ \mathsf{x})\ \Theta\ \ \{\,\mathsf{e}\,\}$$

- t is the result type
- t' is the type of the formal parameter x

Method Declaration

$$\mathsf{t}\ \mathsf{m}\ (\mathsf{t}'\ \mathsf{x})\ \Theta\ \ \{\ \mathsf{e}\ \}$$

- t is the result type
- t' is the type of the formal parameter x
- e is the method's body

Method Declaration

$$\mathsf{t}\ \mathsf{m}\ (\mathsf{t}'\ \mathsf{x})\ \Theta\ \ \{\ \mathsf{e}\ \}$$

- t is the result type
- t' is the type of the formal parameter x
- e is the method's body
- Θ is the effect: a pair $\langle\phi, \psi\rangle$

Method Declaration

$$\mathsf{t}\ \mathsf{m}\ (\mathsf{t}'\ \mathsf{x})\ \Theta\ \ \{\,\mathsf{e}\,\}$$

- t is the result type
- t' is the type of the formal parameter x
- e is the method's body
- Θ is the effect: a pair $\langle \phi, \psi \rangle$
  - ϕ, the re-classification effect, is a set of root classes whose objects could be re-classified during the evaluation of e

Method Declaration

$$\mathsf{t\ m\ (t'\ x)\ \Theta\ \ \{\, e\, \}}$$

- t is the result type
- $\mathsf{t'}$ is the type of the formal parameter x
- e is the method’s body
- Θ is the effect: a pair $\langle \phi, \psi \rangle$
  - ϕ, the re-classification effect, is a set of root classes whose objects could be re-classified during the evaluation of e
  - ψ, the receive effect, is a set of root classes whose objects could receive a method call during the evaluation of e

Example cont.

```
    root class  Player{ int age;
        void wake()           {...}
        void kissed()                       {...}
    }
  state class Frog extends Player{ Vocal pouch;
    void wake()           {pouch.blow()}
    void kissed()                    {this⇓Prince}
  }
state class Prince extends Player{ Weapon sword;
  void wake()           {sword.swing()}
  void kissed()          {...}
}
```

more

Example cont.

```
      root class  Player{ int age;
            void wake()          {...}
            void kissed()                  {...}
      }
   state class Frog extends Player{ Vocal pouch;
      void wake()          {pouch.blow()}
      void kissed()                  {this⇓Prince}
   }
state class Prince extends Player{ Weapon sword;
   void wake()⟨{ }, {}⟩{sword.swing()}
   void kissed()⟨{}, { }⟩{...}
}
```

more

Example cont.

```
            root class  Player{ int age;
                  void wake()           {...}
                  void kissed()                   {...}
            }
        state class Frog extends Player{ Vocal pouch;
           void wake()⟨{ }, {}⟩{pouch.blow()}
           void kissed()⟨ { Player } , { }⟩{this⇓Prince}
        }
   state class Prince extends Player{ Weapon sword;
      void wake()⟨{ }, {}⟩{sword.swing()}
      void kissed()⟨{}, { }⟩{...}
   }
```

more

Example cont.

```
      root class  Player{ int age;
          void wake()⟨{ }, { }⟩{...}
          void kissed()⟨ { Player } , { }⟩{...}
      }
   state class Frog extends Player{ Vocal pouch;
     void wake()⟨{ }, { }⟩{pouch.blow()}
     void kissed()⟨ { Player } , { }⟩{this⇓Prince}
   }
state class Prince extends Player{ Weapon sword;
  void wake()⟨{ }, { }⟩{sword.swing()}
  void kissed()⟨{}, { }⟩{...}
}
```

more

Contents of the talk

- Introduction and Motivation through examples
- Operational Semantics
- Type System
- Soundness
- Translation

Typing Judgments

$$P, \Gamma \vdash \mathsf{e} : \mathsf{t} \, [\![\, \mathsf{c} \,]\!] \, \Theta$$

more

Typing Judgments

$$P, \Gamma \vdash \mathsf{e} : \mathsf{t} \,[\![\, \mathsf{c} \,]\!]\, \Theta$$

- P (the program) contains the class definitions;

more

Typing Judgments

$$P, \Gamma \vdash \texttt{e} : \texttt{t} \,[\![\, \texttt{c} \,]\!]\, \Theta$$

- P (the program) contains the class definitions;
- Γ gives the class of `this` (before the evaluation of `e`) and the type of the formal parameter

more

Typing Judgments

$$P, \Gamma \vdash \mathtt{e} : \mathtt{t} \,[\![\, \mathtt{c} \,]\!]\, \Theta$$

- P (the program) contains the class definitions;
- Γ gives the class of `this` (before the evaluation of `e`) and the type of the formal parameter
- `t` is the type of values returned by the evaluation of `e`

more

Typing Judgments

$$P, \Gamma \vdash \texttt{e} : \texttt{t} \,[\![\, \texttt{c} \,]\!]\, \Theta$$

- P (the program) contains the class definitions;
- Γ gives the class of `this` (before the evaluation of `e`) and the type of the formal parameter
- `t` is the type of values returned by the evaluation of `e`
- `c` is the class of `this` after the evaluation of `e`

more

Typing Judgments

$$P, \Gamma \vdash \texttt{e} : \texttt{t} \,\|\, \texttt{c} \,\|\, \Theta$$

- P (the program) contains the class definitions;
- Γ gives the class of `this` (before the evaluation of `e`) and the type of the formal parameter
- `t` is the type of values returned by the evaluation of `e`
- `c` is the class of `this` after the evaluation of `e`
- Θ is the effect of the evaluation of `e`

more

Typing Judgments

$$P, \Gamma \vdash \mathtt{e} : \mathtt{t} \,\|\, \mathtt{c} \,\|\, \Theta$$

- P (the program) contains the class definitions;
- Γ gives the class of `this` (before the evaluation of `e`) and the type of the formal parameter
- `t` is the type of values returned by the evaluation of `e`
- `c` is the class of `this` after the evaluation of `e`
- Θ is the effect of the evaluation of `e` : a pair $\langle \phi, \psi \rangle$
  - ϕ, the re-classification effect, is a set of root classes whose objects could be re-classified during the evaluation of `e`
  - ψ, the receive effect, is a set of root classes whose objects could receive a method call during the evaluation of `e`

more

Contents of the talk

- Introduction and Motivation through examples
- Operational Semantics
- Type System
- Soundness
- Translation

Soundness

Subject Reduction: if a well-typed configuration reduces to another configuration then the new configuration is well-typed too (no message-not-understood errors)

more

Soundness

Subject Reduction: if a well-typed configuration reduces to another configuration then the new configuration is well-typed too (no message-not-understood errors)

Progress: a well-typed configuration either is a final one or it reduces (no deadlock)

more

Contents of the talk

- Introduction and Motivation through examples
- Operational Semantics
- Type System
- Soundness
- Translation

Translation into Java

- the re-classification was already translated (Ancona, Anderson, Damiani, Drossopoulou, Giannini, Zucca)

Translation into Java

- the re-classification was already translated (Ancona, Anderson, Damiani, Drossopoulou, Giannini, Zucca)
- a class Spawn which extends Thread:
  - for each $\mathbf{spawn}(\mathsf{e})$ a fresh class SpawnLabel which extends Spawn with fields recording effects and where the translation of e is the body of the run method;
  - the translation of $\mathbf{spawn}(\mathsf{e})$ is $\mathbf{new}\ \mathsf{SpawnLabel}(\mathsf{x}).\mathsf{start}();$ `true` ;

Translation into Java

- the re-classification was already translated (Ancona, Anderson, Damiani, Drossopoulou, Giannini, Zucca)
- a class Spawn which extends Thread:
  - for each $\mathbf{spawn}(\mathtt{e})$ a fresh class SpawnLabel which extends Spawn with fields recording effects and where the translation of e is the body of the run method;
  - the translation of $\mathbf{spawn}(\mathtt{e})$ is $\mathbf{new}\ \mathsf{SpawnLabel}(\mathsf{x}).\mathsf{start}();\mathtt{true}$;
- a class Gamma monitoring objects by means of synchronised methods:
  - each method call waits until it can look the required objects;
  - each method return notifies objects unlocks.

Configuration

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, \ldots\} \gg$$

Configuration

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, \ldots\} \gg$$

- e is the expression to be evaluated

Configuration

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, \ldots\} \gg$$

- e is the expression to be evaluated
- χ is the heap

Configuration

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, \ldots\} \gg$$

- e is the expression to be evaluated
- χ is the heap
- ρ is the frame

Configuration

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, \ldots\} \gg$$

- e is the expression to be evaluated
- χ is the heap
- ρ is the frame
- Θ is the effect: a pair $\langle \phi, \psi \rangle$

Configuration

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, \ldots\} \gg$$

- e is the expression to be evaluated
- χ is the heap
- ρ is the frame
- Θ is the effect: a pair $\langle \phi, \psi \rangle$
  - ϕ, the re-classification effect, is a set of root classes whose objects could be re-classified during the evaluation of e
  - ψ, the receive effect, is a set of classes whose objects could receive a method call during the evaluation of e

Configuration

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, \ldots\} \gg$$

Configuration

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, ...\} \gg$$

- γ is the global object state: Addresses $\rightarrow \{-1, 0, 1, 2, ...\}$

Configuration

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, \ldots\} \gg$$

- γ is the global object state: Addresses $\rightarrow \{-1, 0, 1, 2, \ldots\}$
  - $\gamma(\iota) = -1$ means that the object at address ι could be re-classified
  - $\gamma(\iota) \geq 0$ means that $\gamma(\iota)$ threads could use the object at address ι as receiver.

Configuration

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, ...\} \gg$$

- γ is the global object state: Addresses $\rightarrow \{-1, 0, 1, 2, ...\}$
  - $\gamma(\iota) = -1$ means that the object at address ι could be re-classified
  - $\gamma(\iota) \geq 0$ means that $\gamma(\iota)$ threads could use the object at address ι as receiver.
- λ is the local object state: Addresses $\rightarrow \{-1, 0, 1\}$

Configuration

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, ...\} \gg$$

- γ is the global object state: Addresses $\rightarrow \{-1, 0, 1, 2, ...\}$
  - $\gamma(\iota) = -1$ means that the object at address ι could be re-classified
  - $\gamma(\iota) \geq 0$ means that $\gamma(\iota)$ threads could use the object at address ι as receiver.
- λ is the local object state: Addresses $\rightarrow \{-1, 0, 1\}$
  - $\lambda(\iota) = -1$ means that the object at address ι could be re-classified by the current thread
  - $\lambda(\iota) = 0$ means that the current thread does not use the object at address ι as receiver
  - $\lambda(\iota) = 1$ means that the current thread uses the object at address ι as receiver.

Top Level Method Call

$$\ll \chi, \gamma, \{ ..., \langle \rho, \lambda_0, \langle \{\}, \{\} \rangle, \iota.\mathsf{m}(\mathsf{v}) \rangle, ... \} \gg$$

Top Level Method Call

$\chi(\iota) = [\![\ldots]\!]^{\mathsf{c}}$

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda_0, \langle \{\}, \{\} \rangle, \iota.\mathsf{m}(\mathsf{v}) \rangle, \ldots\} \gg$$

Top Level Method Call

$$\chi(\iota) = [[\ldots]]^{\mathsf{c}}$$

$$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\,\mathsf{m}\ (\mathsf{t}'\,\mathsf{x})\ \langle \phi, \psi \rangle\ \ \{\,\mathsf{e}\,\}$$

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda_0, \langle \{\}, \{\} \rangle, \iota.\mathsf{m}(\mathsf{v}) \rangle, \ldots\} \gg$$

Top Level Method Call

$\chi(\iota) = [[...]]^{\mathsf{c}}$

$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\,\mathsf{m}\ (\mathsf{t}'\,\mathsf{x})\ \langle \phi, \psi \rangle\ \ \{\,\mathsf{e}\,\}$

γ is 0 for all objects belonging to classes in ϕ

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda_0, \langle \{\}, \{\} \rangle, \iota.\mathsf{m}(\mathsf{v}) \rangle, ...\} \gg$$

Top Level Method Call

$\chi(\iota) = [[...]]^{\mathsf{c}}$

$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\,\mathsf{m}\ (\mathsf{t}'\,\mathsf{x})\ \langle \phi, \psi \rangle\ \ \{\,\mathsf{e}\,\}$

γ is 0 for all objects belonging to classes in ϕ

γ is ≥ 0 for all objects belonging to classes in $\psi \cup \{\mathsf{c}\}$

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda_0, \langle \{\}, \{\} \rangle, \iota.\mathsf{m}(\mathsf{v}) \rangle, ...\} \gg$$

Top Level Method Call

$\chi(\iota) = [[\ldots]]^{\mathsf{c}}$
$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\ \mathsf{m}\ (\mathsf{t}'\ \mathsf{x})\ \langle \phi, \psi \rangle\ \ \{\,\mathsf{e}\,\}$
γ is 0 for all objects belonging to classes in ϕ
γ is ≥ 0 for all objects belonging to classes in $\psi \cup \{\mathsf{c}\}$

$$\begin{array}{c} \ll \chi, \gamma, \{\ldots, \langle \rho, \lambda_0, \langle \{\}, \{\} \rangle, \iota.\mathsf{m}(\mathsf{v}) \rangle, \ldots\} \gg \\ \big\downarrow P \\ \ll \chi, \gamma', \{\ldots, \langle \rho, \lambda', \langle \phi, \psi \cup \{\mathsf{c}\} \rangle, \mathbf{return}^{\mathsf{o}}(\rho', \mathsf{e}) \rangle, \ldots\} \gg \end{array}$$

Top Level Method Call

$\chi(\iota) = [[\ldots]]^{\mathsf{c}}$

$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\,\mathsf{m}\ (\mathsf{t}'\,\mathsf{x})\ \langle \phi, \psi \rangle\ \ \{\,\mathsf{e}\,\}$

γ is 0 for all objects belonging to classes in ϕ

γ is ≥ 0 for all objects belonging to classes in $\psi \cup \{\mathsf{c}\}$

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda_0, \langle \{\}, \{\} \rangle, \iota.\mathsf{m}(\mathsf{v}) \rangle, \ldots\} \gg$$

$$\Big\downarrow P$$

$$\ll \chi, \gamma', \{\ldots, \langle \rho, \lambda', \langle \phi, \psi \cup \{\mathsf{c}\} \rangle, \mathbf{return}^{\mathsf{o}}(\rho', \mathsf{e}) \rangle, \ldots\} \gg$$

$\rho' = [\mathsf{x} \mapsto \mathsf{v}, \mathtt{this} \mapsto \iota]$

Top Level Method Call

$\chi(\iota) = [[\ldots]]^{\mathsf{c}}$
$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\,\mathsf{m}\ (\mathsf{t}'\,\mathsf{x})\ \langle\phi, \psi\rangle\ \ \{\,\mathsf{e}\,\}$
γ is 0 for all objects belonging to classes in ϕ
γ is ≥ 0 for all objects belonging to classes in $\psi \cup \{\mathsf{c}\}$

$$\ll \chi, \gamma, \{\ldots, \langle\rho, \lambda_0, \langle\{\}, \{\}\rangle, \iota.\mathsf{m}(\mathsf{v})\rangle, \ldots\} \gg$$
$$\downarrow P$$
$$\ll \chi, \gamma', \{\ldots, \langle\rho, \lambda', \langle\phi, \psi \cup \{\mathsf{c}\}\rangle, \mathbf{return}^{\mathsf{o}}(\rho', \mathsf{e})\rangle, \ldots\} \gg$$

$\rho' = [\mathsf{x} \mapsto \mathsf{v}, \mathtt{this} \mapsto \iota]$
γ' is -1 for all objects belonging to classes in ϕ

Top Level Method Call

$\chi(\iota) = [[\ldots]]^{\mathsf{c}}$
$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\ \mathsf{m}\ (\mathsf{t}'\ \mathsf{x})\ \langle \phi, \psi \rangle\ \ \{\ \mathsf{e}\ \}$
γ is 0 for all objects belonging to classes in ϕ
γ is ≥ 0 for all objects belonging to classes in $\psi \cup \{\mathsf{c}\}$

$$\begin{gathered} \ll \chi, \gamma, \{\ldots, \langle \rho, \lambda_0, \langle \{\}, \{\} \rangle, \iota.\mathsf{m}(\mathsf{v}) \rangle, \ldots\} \gg \\ \big\downarrow P \\ \ll \chi, \gamma', \{\ldots, \langle \rho, \lambda', \langle \phi, \psi \cup \{\mathsf{c}\} \rangle, \mathbf{return}^{\mathsf{o}}(\rho', \mathsf{e}) \rangle, \ldots\} \gg \end{gathered}$$

$\rho' = [\mathsf{x} \mapsto \mathsf{v}, \mathtt{this} \mapsto \iota]$
γ' is -1 for all objects belonging to classes in ϕ
γ' is $\gamma + 1$ for all objects belonging to classes in $\psi \cup \{\mathsf{c}\}$

Top Level Method Call

$\chi(\iota) = [[\ldots]]^{\mathsf{c}}$
$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\,\mathsf{m}\ (\mathsf{t}'\,\mathsf{x})\ \langle \phi, \psi \rangle\ \ \{\,\mathsf{e}\,\}$
γ is 0 for all objects belonging to classes in ϕ
γ is ≥ 0 for all objects belonging to classes in $\psi \cup \{\mathsf{c}\}$

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda_0, \langle \{\}, \{\} \rangle, \iota.\mathsf{m}(\mathsf{v}) \rangle, \ldots\} \gg$$
$$\Big\downarrow P$$
$$\ll \chi, \gamma', \{\ldots, \langle \rho, \lambda', \langle \phi, \psi \cup \{\mathsf{c}\} \rangle, \mathbf{return}^{\mathsf{o}}(\rho', \mathsf{e}) \rangle, \ldots\} \gg$$

$\rho' = [\mathsf{x} \mapsto \mathsf{v}, \mathtt{this} \mapsto \iota]$
γ' is -1 for all objects belonging to classes in ϕ
γ' is $\gamma + 1$ for all objects belonging to classes in $\psi \cup \{\mathsf{c}\}$
λ' is -1 for all objects belonging to classes in ϕ

Top Level Method Call

$\chi(\iota) = [[\ldots]]^{\mathsf{c}}$
$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\,\mathsf{m}\ (\mathsf{t}'\,\mathsf{x})\ \langle\phi, \psi\rangle\ \ \{\,\mathsf{e}\,\}$
γ is 0 for all objects belonging to classes in ϕ
γ is ≥ 0 for all objects belonging to classes in $\psi \cup \{\mathsf{c}\}$

$$\begin{array}{c}
\ll \chi, \gamma, \{\ldots, \langle\rho, \lambda_0, \langle\{\}, \{\}\rangle, \iota.\mathsf{m}(\mathsf{v})\rangle, \ldots\} \gg \\
\Big\downarrow P \\
\ll \chi, \gamma', \{\ldots, \langle\rho, \lambda', \langle\phi, \psi \cup \{\mathsf{c}\}\rangle, \mathbf{return}^{\mathsf{o}}(\rho', \mathsf{e})\rangle, \ldots\} \gg
\end{array}$$

$\rho' = [\mathsf{x} \mapsto \mathsf{v}, \mathtt{this} \mapsto \iota]$
γ' is -1 for all objects belonging to classes in ϕ
γ' is $\gamma + 1$ for all objects belonging to classes in $\psi \cup \{\mathsf{c}\}$
λ' is -1 for all objects belonging to classes in ϕ
λ' is 1 for all objects belonging to classes in $\psi \cup \{\mathsf{c}\}$

Inner Method Call

$$\ll \chi, \gamma, \{\ldots, \langle\rho, \lambda, \Theta, \iota.\mathsf{m}(\mathsf{v})\rangle, \ldots\} \gg$$

Inner Method Call

$\chi(\iota) = [\![\ldots]\!]^{\mathsf{c}}$

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \iota.\mathsf{m}(\mathsf{v})\rangle, \ldots\} \gg$$

Inner Method Call

$$\chi(\iota) = [[\ldots]]^{\mathsf{c}}$$
$$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\,\mathsf{m}\ (\mathsf{t}'\,\mathsf{x})\ \langle\phi, \psi\rangle\ \ \{\,\mathsf{e}\,\}$$

$$\ll \chi, \gamma, \{\ldots, \langle\rho, \lambda, \Theta, \iota.\mathsf{m}(\mathsf{v})\rangle, \ldots\} \gg$$

Inner Method Call

$\chi(\iota) = [[\ldots]]^{\mathsf{c}}$
$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\,\mathsf{m}\ (\mathsf{t}'\,\mathsf{x})\ \langle \phi, \psi \rangle\ \ \{\,\mathsf{e}\,\}$
$\Theta \neq \langle \{\}, \{\} \rangle$

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \iota.\mathsf{m}(\mathsf{v}) \rangle, \ldots\} \gg$$

Inner Method Call

$\chi(\iota) = [[\ldots]]^{\mathsf{c}}$
$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\,\mathsf{m}\ (\mathsf{t}'\,\mathsf{x})\ \langle \phi, \psi \rangle\ \ \{\, \mathsf{e} \,\}$
$\Theta \neq \langle \{\}, \{\} \rangle$

$$\ll \chi, \gamma, \{ \ldots, \langle \rho, \lambda, \Theta, \iota.\mathsf{m}(\mathsf{v}) \rangle, \ldots \} \gg$$
$$\Big\downarrow P$$
$$\ll \chi, \gamma, \{ \ldots, \langle \rho, \lambda, \Theta, \mathbf{return}^{\mathsf{i}}(\rho', \mathsf{e}) \rangle, \ldots \} \gg$$

Inner Method Call

$\chi(\iota) = [[\ldots]]^{\mathsf{c}}$
$\mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\,\mathsf{m}\ (\mathsf{t}'\,\mathsf{x})\ \langle \phi, \psi \rangle\ \ \{\,\mathsf{e}\,\}$
$\Theta \neq \langle \{\}, \{\} \rangle$

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \iota.\mathsf{m}(\mathsf{v}) \rangle, \ldots\} \gg \Big\downarrow P \ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathbf{return}^{\mathsf{i}}(\rho', \mathsf{e}) \rangle, \ldots\} \gg$$

$\rho' = [\mathsf{x} \mapsto \mathsf{v}, \mathtt{this} \mapsto \iota]$

Spawn

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathcal{C}[\mathbf{spawn}(\mathsf{e})]\rangle, ...\} \gg$$

Spawn

$\mathcal{C}$ does not contain $\mathbf{return}^{\eta}(...,...)$

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathcal{C}[\mathbf{spawn}(\mathsf{e})]\rangle, ...\} \gg$$

Spawn

$\mathcal{C}$ does not contain $\mathbf{return}^{\eta}(...,...)$

$$\begin{array}{c}\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathcal{C}[\mathbf{spawn}(\mathsf{e})]\rangle, ...\} \gg \\ \big\downarrow P \\ \ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathcal{C}[\mathsf{true}]\rangle, \langle \rho, \lambda_0, \Theta_0, \mathsf{e}\rangle, ...\} \gg\end{array}$$

Spawn

$\mathcal{C}$ does not contain $\mathbf{return}^{\eta}(...,...)$

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathcal{C}[\mathbf{spawn}(\mathsf{e})]\rangle, ...\} \gg$$

$$\big\downarrow P$$

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathcal{C}[\mathsf{true}]\rangle, \langle \rho, \lambda_0, \Theta_0, \mathsf{e}\rangle, ...\} \gg$$

$\Theta_0 = \langle \{\}, \{\} \rangle$

Spawn

$\mathcal{C}$ does not contain $\mathbf{return}^{\eta}(...,...)$

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathcal{C}[\mathbf{spawn}(\mathsf{e})]\rangle, ...\} \gg$$
$$\downarrow P$$
$$\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathcal{C}[\mathsf{true}]\rangle, \langle \rho, \lambda_0, \Theta_0, \mathsf{e}\rangle, ...\} \gg$$

$\Theta_0 = \langle \{\}, \{\} \rangle$

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathcal{E}[\mathbf{return}^{\eta}(\rho', \mathcal{C}[\mathbf{spawn}(\mathsf{e})])]\rangle, ...\} \gg$$

back

Spawn

$\mathcal{C}$ does not contain $\mathbf{return}^{\eta}(\ldots, \ldots)$

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathcal{C}[\mathbf{spawn}(\mathsf{e})]\rangle, \ldots\} \gg$$

$$\Big\downarrow P$$

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathcal{C}[\mathsf{true}]\rangle, \langle \rho, \lambda_0, \Theta_0, \mathsf{e}\rangle, \ldots\} \gg$$

$\Theta_0 = \langle \{\}, \{\} \rangle$

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathcal{E}[\mathbf{return}^{\eta}(\rho', \mathcal{C}[\mathbf{spawn}(\mathsf{e})])]\rangle, \ldots\} \gg$$

$$\Big\downarrow P$$

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathcal{E}[\mathbf{return}^{\eta}(\rho', \mathcal{C}[\mathsf{true}])]\rangle, \langle \rho', \lambda_0, \Theta_0, \mathsf{e}\rangle, \ldots\} \gg$$

back

Typing Rules

$$\frac{\begin{array}{l}\mathsf{c}' = \text{root-superclass of } \mathsf{c} \\ \quad = \text{ root-superclass of } \Gamma(\mathtt{this})\end{array}}{P, \Gamma \vdash \mathtt{this}{\Downarrow}\mathtt{c} : \mathtt{c} \parallel \mathsf{c} \parallel \langle \{\mathsf{c}'\}, \{\} \rangle} \quad (recl)$$

back

Typing Rules

$$\frac{\begin{array}{l}\mathsf{c}' = \text{root-superclass of } \mathsf{c} \\ \quad = \text{ root-superclass of } \Gamma(\mathtt{this})\end{array}}{P, \Gamma \vdash \mathtt{this}\Downarrow\mathsf{c} \ : \ \mathsf{c} \parallel \mathsf{c} \parallel \langle\{\mathsf{c}'\}, \{\}\rangle} \quad (recl)$$

$$\frac{P, \{\mathsf{t}_1 \ \mathsf{x}, \mathsf{Object} \ \mathtt{this}\} \vdash \mathsf{e} \ : \ \mathsf{t} \parallel \mathsf{Object} \parallel \Theta}{P, \{\mathsf{t}_1 \ \mathsf{x}, \mathsf{c} \ \mathtt{this}\} \vdash \mathbf{spawn}(\mathsf{e}) \ : \ \mathsf{bool} \parallel \mathsf{c} \parallel \langle\{\}, \{\}\rangle} \quad (spawn)$$

back

Typing Rules

$$\frac{\begin{array}{l} \mathsf{c}' = \text{root-superclass of } \mathsf{c} \\ \quad = \text{ root-superclass of } \Gamma(\texttt{this}) \end{array}}{P, \Gamma \vdash \texttt{this}{\Downarrow}\mathsf{c} : \mathsf{c} \parallel \mathsf{c} \parallel \langle \{\mathsf{c}'\}, \{\} \rangle} \quad (\mathit{recl})$$

$$\frac{P, \{\mathsf{t}_1\ \mathsf{x}, \mathsf{Object}\ \texttt{this}\} \vdash \mathsf{e} : \mathsf{t} \parallel \mathsf{Object} \parallel \Theta}{P, \{\mathsf{t}_1\ \mathsf{x}, \mathsf{c}\ \texttt{this}\} \vdash \mathbf{spawn}(\mathsf{e}) : \mathsf{bool} \parallel \mathsf{c} \parallel \langle \{\}, \{\} \rangle} \quad (\mathit{spawn})$$

$$\frac{\begin{array}{l} P, \Gamma \vdash \mathsf{e}_0 : \mathsf{c} \parallel \Gamma(\texttt{this}) \parallel \langle \phi_0, \psi_0 \rangle \\ P, \Gamma \vdash \mathsf{e}_1 : \mathsf{t}_1 \parallel \Gamma(\texttt{this}) \parallel \langle \phi_1, \psi_1 \rangle \\ \text{root-superclass of } \Gamma(\texttt{this}) \notin \phi_0 \cup \phi_1 \\ \mathcal{M}(P, \mathsf{c}, \mathsf{m}) = \mathsf{t}\ \mathsf{m}(\mathsf{t}_1\ \mathsf{x})\ \langle \phi, \psi \rangle\ \{\ \ldots\ \} \end{array}}{\begin{array}{c} P, \Gamma \vdash \mathsf{e}_0.\mathsf{m}(\mathsf{e}_1) : \mathsf{t} \parallel \phi @_P \Gamma(\texttt{this}) \parallel \Theta \\ \Theta = \langle \phi, \psi \rangle \cup \langle \phi_0, \psi_0 \rangle \cup \langle \phi_1, \psi_1 \rangle \cup \langle \{\}, \{\mathsf{c}\} \rangle \end{array}} \quad (\mathit{meth})$$

back

Subject Reduction

$$\begin{tikzcd} \ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, \ldots\} \gg \arrow[d, "P"] \\ \ll \chi', \gamma', \{\ldots, \langle \rho', \lambda', \Theta', \mathsf{e}' \rangle, \ldots\} \gg \end{tikzcd}$$

$$P, \Gamma \vdash \mathsf{e} : \mathsf{t} \parallel \mathsf{c} \parallel \Theta''$$

Subject Reduction

$$\begin{tikzcd}\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathsf{e} \rangle, \ldots\} \gg \arrow[d, "P"] \\ \ll \chi', \gamma', \{\ldots, \langle \rho', \lambda', \Theta', \mathsf{e}' \rangle, \ldots\} \gg\end{tikzcd}$$

$$P, \Gamma \ \vdash \ \mathsf{e} \ : \ \mathsf{t} \parallel \mathsf{c} \parallel \Theta''$$

$$\Downarrow$$

$$P, \Gamma' \ \vdash \ \mathsf{e}' \ : \ \mathsf{t} \parallel \mathsf{c} \parallel \Theta''$$

Subject Reduction cont.

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathcal{E}[\mathbf{spawn}(\mathsf{e})]\rangle, ...\} \gg$$

$$\downarrow P$$

$$\ll \chi, \gamma, \{..., \langle \rho, \lambda, \Theta, \mathcal{E}[\mathsf{true}]\rangle, \langle \rho', \lambda_0, \Theta_0, \mathsf{e}\rangle, ...\} \gg$$

$$P, \Gamma \ \vdash \ \mathcal{E}[\mathbf{spawn}(\mathsf{e})] \ : \ \mathsf{t} \parallel \mathsf{c} \parallel \Theta''$$

Subject Reduction cont.

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathcal{E}[\mathbf{spawn}(\mathsf{e})]\rangle, \ldots\} \gg$$

$$\downarrow P$$

$$\ll \chi, \gamma, \{\ldots, \langle \rho, \lambda, \Theta, \mathcal{E}[\mathsf{true}]\rangle, \langle \rho', \lambda_0, \Theta_0, \mathsf{e}\rangle, \ldots\} \gg$$

$$P, \Gamma \ \vdash \ \mathcal{E}[\mathbf{spawn}(\mathsf{e})] \ : \ \mathsf{t} \parallel \mathsf{c} \parallel \Theta''$$

$$\Downarrow$$

$$P, \Gamma \ \vdash \ \mathcal{E}[\mathsf{true}] \ : \ \mathsf{t} \parallel \mathsf{c} \parallel \Theta''$$

$$P, \Gamma' \ \vdash \ \mathsf{e} \ : \ \mathsf{t}' \parallel \mathsf{Object} \parallel \Theta'''$$

$$\Gamma' = \{\Gamma(\mathsf{x})\ \mathsf{x}, \mathsf{Object}\ \mathtt{this}\}$$

Progress

e_0 is a well-typed expression

$$\ll \chi_0, \gamma_0, \{\langle \rho_0, \lambda_0, \Theta_0, \mathsf{e}_0 \rangle\} \gg$$

$$\downarrow P$$

$$\vdots$$

$$\downarrow P$$

$$\ll \chi, \gamma, \{\langle \rho_1, \lambda_1, \Theta_1, \mathsf{e}_1 \rangle, \ldots, \langle \rho_\mathsf{n}, \lambda_\mathsf{n}, \Theta_\mathsf{n}, \mathsf{e}_\mathsf{n} \rangle\} \gg$$

there is one e_i which is not a value

back

Progress

e_0 is a well-typed expression

$$\ll \chi_0, \gamma_0, \{\langle \rho_0, \lambda_0, \Theta_0, \mathsf{e}_0 \rangle\} \gg$$

$$\downarrow P$$

$$\vdots$$

$$\downarrow P$$

$$\ll \chi, \gamma, \{\langle \rho_1, \lambda_1, \Theta_1, \mathsf{e}_1 \rangle, \ldots, \langle \rho_\mathsf{n}, \lambda_\mathsf{n}, \Theta_\mathsf{n}, \mathsf{e}_\mathsf{n} \rangle\} \gg$$

there is one e_i which is not a value

$$\Downarrow$$

$$\ll \chi, \gamma, \{\langle \rho_1, \lambda_1, \Theta_1, \mathsf{e}_1 \rangle, \ldots, \langle \rho_\mathsf{n}, \lambda_\mathsf{n}, \Theta_\mathsf{n}, \mathsf{e}_\mathsf{n} \rangle\} \gg$$

$$\downarrow P$$

$$\cdots$$

back