Abstracting Effect Systems for Algebraic Effect Handlers

Takuma Yoshioka *1, Taro Sekiyama *2, Atsushi Igarashi *1

*1 Kyoto University, Japan *2 NII & SOKENDAI, Japan

Background:
Algebraic Effect Handlers

- **Algebraic effect handlers** [Plotkin and Pretnar 2009] can represent some computational effects such as mutable states and exceptions
- They have two components, **operations** and **handlers**, which are similar to exception raising and handling, but can resume the computation after operation calls

```
handle
  put 42; get ()
with {
  put -> ...,
  get -> ...
} 0
```

Background:

Algebraic Effect Handlers

- **Algebraic effect handlers** [Plotkin and Pretnar 2009] can represent some computational effects such as mutable states and exceptions
- They have two components, **operations** and **handlers**, which are similar to exception raising and handling, but can resume the computation after operation calls

put/get are operations that write into/read from one mutable cell

```
handle
  put 42; get ()
with {
  put -> ...,
  get -> ...
} 0
```

Background:

Algebraic Effect Handlers

- **Algebraic effect handlers** [Plotkin and Pretnar 2009] can represent some computational effects such as mutable states and exceptions
- They have two components, **operations** and **handlers**, which are similar to exception raising and handling, but can resume the computation after operation calls

put/get are operations that write into/read from one mutable cell

This handler implements the behavior of put and get

```
handle
  put 42; get ()
with {
  put -> ...,
  get -> ...
} 0
```

Background:

Algebraic Effect Handlers

- **Algebraic effect handlers** [Plotkin and Pretnar 2009] can represent some computational effects such as mutable states and exceptions
- They have two components, **operations** and **handlers**, which are similar to exception raising and handling, but can resume the computation after operation calls

put/get are operations that write into/read from one mutable cell

This handler implements the behavior of put and get

```
handle
  put 42; get ()
with {
  put -> ...,
  get -> ...
} 0
```

→ 42

Background:
Unhandled Operations

- **Unhandled operations** cause runtime errors

There is no handler for put/get

```
put 42; get ()
```

→ **gets stuck!**

- Their behavior are similar to uncaught exceptions

Background:
Effect Systems

- **Effect systems** for algebraic effect handlers statically track unhandled operations as **effects ε**, and guarantee that every operation is handled appropriately
- E.g., an effect system that uses **sets** to represent effects ε

$$\Gamma \vdash e : A \mid \varepsilon$$

$$\frac{\ldots \qquad \Gamma \vdash \texttt{put 42; get ()} : \texttt{int} \mid \{\texttt{put}, \texttt{get}\}}{\Gamma \vdash \texttt{handle put 42; get () with h} : \texttt{int -> int} \mid \{\}}$$

Background:

Effect Systems

- **Effect systems** for algebraic effect handlers statically track unhandled operations as **effects ε**, and guarantee that every operation is handled appropriately
- E.g., an effect system that uses **sets** to represent effects ε

```
Γ ⊢ e : A | ε
```

means operations put and get are not yet handled

```
   ...    Γ ⊢ put 42; get () : int | {put, get}
──────────────────────────────────────────────────
Γ ⊢ handle put 42; get () with h : int -> int | {}
```

Background:
Effect Systems

- **Effect systems** for algebraic effect handlers statically track unhandled operations as **effects ε**, and guarantee that every operation is handled appropriately
- E.g., an effect system that uses **sets** to represent effects ε

```
Γ ⊢ e : A | ε
```

means operations put and get are not yet handled

```
       ...      Γ ⊢ put 42; get () : int | {put, get}
─────────────────────────────────────────────────────
 Γ ⊢ handle put 42; get () with h : int -> int | {}
```

The empty set means every operation is handled

Background:

Variations of Effect Systems

- Various **effect representations**, such as sets, have been studied
  - These various implementations of effects are motivated by, e.g., type inference and expressive power

effect representations	sets	simple rows	scoped rows
language example	Eff [Pretnar+ 2013]	Links [Hillerström+ 2016]	Koka [Leijen 2017]

Simple (resp. scoped) rows are sequences of operations without (resp. with) duplicating the same operations in one effect

Our Research Question

- What are essential commonalities/differences among various effect representations?
  - This unknown relationship suggests that there is no guide for language designers who will create a new style

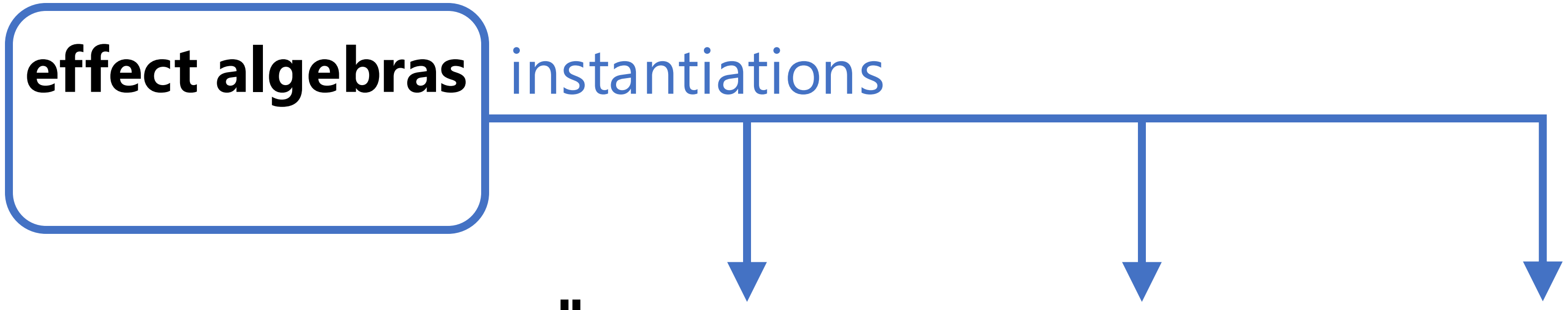
effect representations	sets	simple rows	scoped rows
language example	Eff [Pretnar+ 2013]	Links [Hillerström+ 2016]	Koka [Leijen 2017]

Simple (resp. scoped) rows are sequences of operations without (resp. with) duplicating the same operations in one effect

Our Approach:
Abstracting Effect Representations

Our Approach:

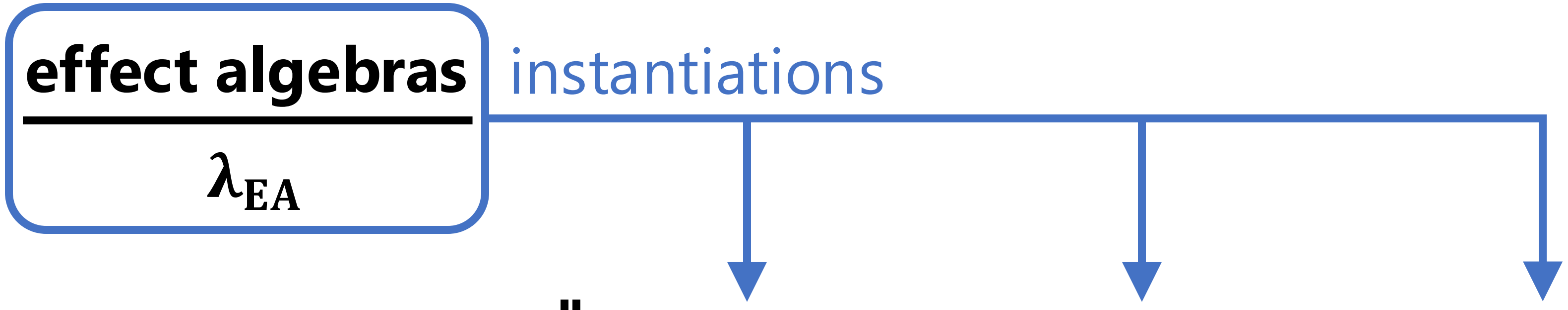
Abstracting Effect Representations



effect representations	sets	simple rows	scoped rows
language example	Eff [Pretnar+ 2013]	Links [Hillerström+ 2016]	Koka [Leijen 2017]

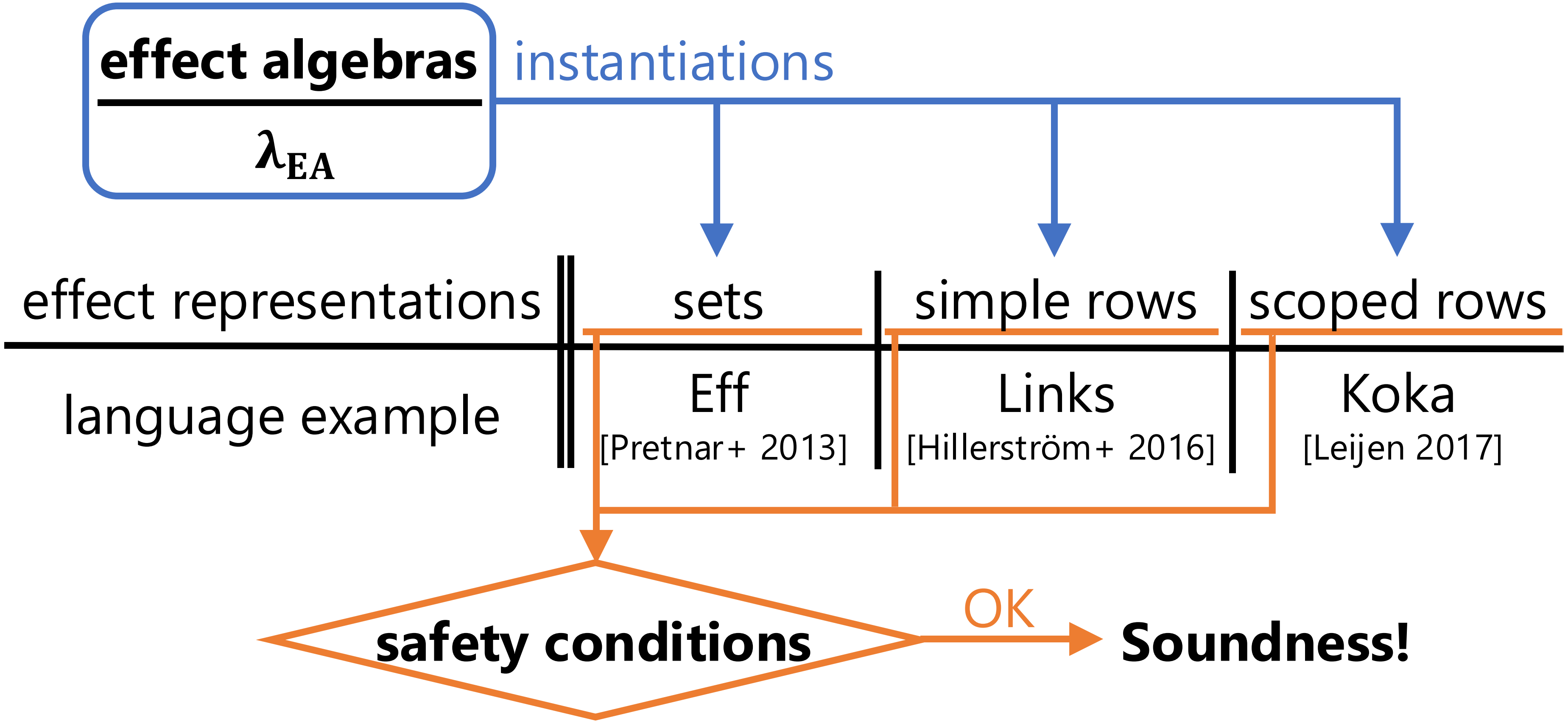
Our Approach:

Abstracting Effect Representations

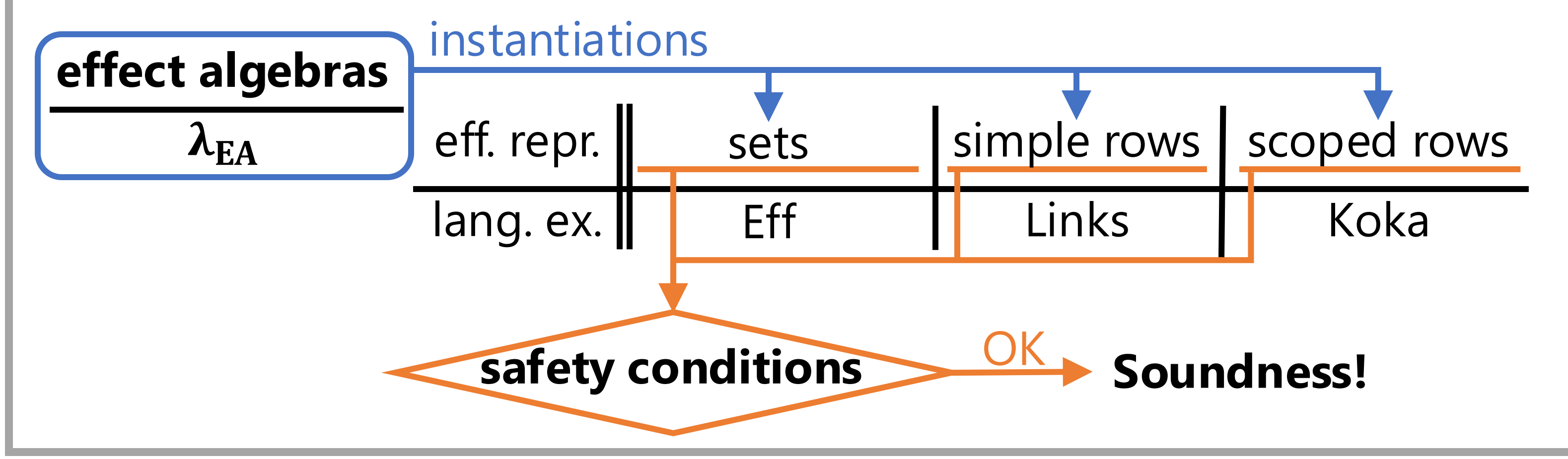


effect representations	sets	simple rows	scoped rows
language example	Eff [Pretnar+ 2013]	Links [Hillerström+ 2016]	Koka [Leijen 2017]

Our Approach:
Abstracting Effect Representations



Contributions



- **Effect algebras**
  - Abstraction of effect representations and their manipulations
- λ_{EA}
  - A λ-calculus with effect handlers, **parameterized over effect algebras**
- **Safety conditions**
  - Requirements on effect algebras for soundness
- **Main Theorem:**
  any instance of λ_{EA} is sound if it meets the safety conditions

Outline

- **Effect algebras: abstraction of effect representations and their manipulations**
  - Manipulations of effect representations
  - Definition
  - Abstracted typing rules
- λ_{EA}: λ-calculus Parameterized over Effect Algebras
- Safety conditions and soundness
- Summary and other topics

Background:

Manipulations of Effect Representations

- Below are the manipulations of a set representation

$$\frac{\texttt{...} \qquad \texttt{op : A -> B} \text{ is given}}{\Gamma \vdash \texttt{op v : B} \mid \{\texttt{op}\}} \text{[Op]}$$

injects the operation op to the singleton set {op}

subeffecting is inclusion

$$\frac{\Gamma \vdash \texttt{e : A} \mid \varepsilon \qquad \varepsilon \subseteq \varepsilon'}{\Gamma \vdash \texttt{e : A} \mid \varepsilon'} \text{[SubEff]}$$

removes the operation op from ε

$$\frac{\texttt{...} \qquad \Gamma \vdash \texttt{e : A} \mid \varepsilon \qquad \{\texttt{op}\} \cup \varepsilon' = \varepsilon}{\Gamma \vdash \texttt{handle e with \{op -> ...\} : B} \mid \varepsilon'} \text{[Handle]}$$

Our Contribution:
Effect Algebras

- **An effect algebra** is a tuple $\langle O, E, (-)^{\uparrow}, \odot, \mathbb{0} \rangle$
  - O is a set of operations
  - E is a set of effects
  - $(-)^{\uparrow} : O \to E$
  - $\langle E, \odot, \mathbb{0} \rangle$ forms a partial monoid
- E.g., an effect algebra for a set repr. is $\langle O, 2^O, \{-\}, \cup, \{\,\} \rangle$
- Remark: our notations and basic idea are based on row algebras [Morris and McKinna 2019]

Our Contribution:
Effect Algebras

operation injection

- **An effect algebra** is a tuple $\langle O, E, (-)^{\uparrow}, \odot, \mathbb{0} \rangle$
  - O is a set of operations
  - E is a set of effects
  - $(-)^{\uparrow} : O \to E$
  - $\langle E, \odot, \mathbb{0} \rangle$ forms a partial monoid
- E.g., an effect algebra for a set repr. is $\langle O, 2^{O}, \{-\}, \cup, \{\,\} \rangle$
- Remark: our notations and basic idea are based on row algebras [Morris and McKinna 2019]

Our Contribution:
Effect Algebras

- **An effect algebra** is a tuple $\langle O, E, (-)^{\uparrow}, \odot, \mathbb{0} \rangle$
  - O is a set of operations
  - E is a set of effects
  - $(-)^{\uparrow} : O \to E$
  - $\langle E, \odot, \mathbb{0} \rangle$ forms a partial monoid
- E.g., an effect algebra for a set repr. is $\langle O, 2^{O}, \{-\}, \cup, \{\,\} \rangle$
- Remark: our notations and basic idea are based on row algebras [Morris and McKinna 2019]

operation injection

effect concatenation

Our Contribution:
Effect Algebras

operation injection

the empty effect

- **An effect algebra** is a tuple $\langle O, E, (-)^{\uparrow}, \odot, \mathbb{0} \rangle$

effect concatenation

  - O is a set of operations
  - E is a set of effects
  - $(-)^{\uparrow} : O \to E$
  - $\langle E, \odot, \mathbb{0} \rangle$ forms a partial monoid
- E.g., an effect algebra for a set repr. is $\langle O, 2^{O}, \{-\}, \cup, \{\} \rangle$
- Remark: our notations and basic idea are based on row algebras [Morris and McKinna 2019]

Typing Rules based on Effect Algebras

Typing Rules based on Effect Algebras

$$\frac{\ldots \qquad \texttt{op} : \texttt{A} \to \texttt{B} \text{ is given}}{\Gamma \vdash \texttt{op}\ \texttt{v} : \texttt{B} \mid (\texttt{op})^{\uparrow}}\ [\text{Op}]$$

$(-)^{\uparrow}$ **injects** the operation op to the single effect $(\text{op})^{\uparrow}$

Typing Rules based on Effect Algebras

$$\frac{\ldots \qquad \texttt{op} : \texttt{A} \texttt{->} \texttt{B} \text{ is given}}{\Gamma \vdash \texttt{op}\ \texttt{v} : \texttt{B} \mid (\texttt{op})^{\uparrow}}\ [\text{Op}]$$

{op} in the set repr.

$(-)^{\uparrow}$ **injects** the operation op to the single effect $(\text{op})^{\uparrow}$

Typing Rules based on Effect Algebras

$$\frac{\ldots \quad op : A \to B \text{ is given}}{\Gamma \vdash op\ v : B \mid (op)^{\uparrow}}\ [\text{Op}]$$

$(-)^{\uparrow}$ **injects** the operation op to the single effect $(op)^{\uparrow}$

{op} in the set repr.

$$\frac{\Gamma \vdash e : A \mid \varepsilon \qquad \varepsilon \oslash \varepsilon'}{\Gamma \vdash e : A \mid \varepsilon'}\ [\text{SubEff}]$$

subeffecting $\varepsilon \oslash \varepsilon'$ is defined as $\exists \varepsilon''.\varepsilon \odot \varepsilon'' = \varepsilon'$

Typing Rules based on Effect Algebras

$$\frac{\ldots \qquad op : A \to B \text{ is given}}{\Gamma \vdash op\ v : B \mid (op)^{\uparrow}}\ [\text{Op}]$$

$(-)^{\uparrow}$ **injects** the operation op to the single effect $(op)^{\uparrow}$

{op} in the set repr.

$\varepsilon \subseteq \varepsilon'$ in the set repr.

$$\frac{\Gamma \vdash e : A \mid \varepsilon \qquad \varepsilon \lessdot \varepsilon'}{\Gamma \vdash e : A \mid \varepsilon'}\ [\text{SubEff}]$$

subeffecting $\varepsilon \lessdot \varepsilon'$ is defined as $\exists \varepsilon''. \varepsilon \odot \varepsilon'' = \varepsilon'$

Typing Rules based on Effect Algebras

$$\frac{\ldots \qquad \texttt{op : A -> B} \text{ is given}}{\Gamma \vdash \texttt{op v : B} \mid (\texttt{op})^{\uparrow}}\ [\text{Op}]$$

$(-)^{\uparrow}$ **injects** the operation op to the single effect $(\texttt{op})^{\uparrow}$

{op} in the set repr.

$\varepsilon \subseteq \varepsilon'$ in the set repr.

$$\frac{\Gamma \vdash e : A \mid \varepsilon \qquad \varepsilon \lessdot \varepsilon'}{\Gamma \vdash e : A \mid \varepsilon'}\ [\text{SubEff}]$$

subeffecting $\varepsilon \lessdot \varepsilon'$ is defined as $\exists \varepsilon''. \varepsilon \odot \varepsilon'' = \varepsilon'$

removes the operation op from ε

$$\frac{\ldots \qquad \Gamma \vdash e : A \mid \varepsilon \qquad (\texttt{op})^{\uparrow} \odot \varepsilon' = \varepsilon}{\Gamma \vdash \texttt{handle}\ e\ \texttt{with}\ \{\texttt{op -> ...}\} : B \mid \varepsilon'}\ [\text{Handle}]$$

Typing Rules based on Effect Algebras

$$\frac{\ldots \quad op : A \to B \text{ is given}}{\Gamma \vdash op\ v : B \mid (op)^{\uparrow}}\ [\text{Op}]$$

$(-)^{\uparrow}$ **injects** the operation op to the single effect $(op)^{\uparrow}$

{op} in the set repr.

$$\frac{\Gamma \vdash e : A \mid \varepsilon \quad \varepsilon \lessdot \varepsilon'}{\Gamma \vdash e : A \mid \varepsilon'}\ [\text{SubEff}]$$

$\varepsilon \subseteq \varepsilon'$ in the set repr.

subeffecting $\varepsilon \lessdot \varepsilon'$ is defined as $\exists \varepsilon''. \varepsilon \odot \varepsilon'' = \varepsilon'$

$$\frac{\ldots \quad \Gamma \vdash e : A \mid \varepsilon \quad (op)^{\uparrow} \odot \varepsilon' = \varepsilon}{\Gamma \vdash \textbf{handle}\ e\ \textbf{with}\ \{op \to \ldots\} : B \mid \varepsilon'}\ [\text{Handle}]$$

$\{op\} \cup \varepsilon' = \varepsilon$ in the set repr.

removes the operation op from ε

Outline

- Effect algebras: abstraction of eff. repr. and their manipulations
- **$\lambda_{\mathbf{EA}}$: λ-calculus Parameterized over Effect Algebras**
- Safety conditions and soundness
- Summary and other topics

Our Contribution: $\lambda_{\mathbf{EA}}$

- A λ-calculus parameterized over **effect algebras**
- Supporting:
  - Explicit polymorphism over types, operations, and effects
  - Deep effect handlers
- Not supporting:
  - Bounded row polymorphism as in Links [Hillerström+ 2016]
  - Other handling mechanisms
    - Local effects [Biernacki+ 2019]
    - Tunneling [Zhang and Myers 2019]
    - Lexically scoped handlers [Biernacki+ 2020, Brachthäuser+ 2020]

Outline

- Effect algebras: abstraction of eff. repr. and their manipulations
- λ_{EA}: $\lambda-$calculus Parameterized over Effect Algebras
- **Safety conditions and soundness**
  - Safety conditions
  - Main theorem
- Summary and other topics

Our Contribution:

requirements on effect algebras

Safety Conditions for Soundness

- Condition (1): no op satisfies $(\mathsf{op})^{\uparrow} \olessthan \mathbb{0}$
  - Intuition: subeffecting cannot erase information about operation calls
  - Recall that $\mathbb{0}$ represents the empty effect
  - This condition excludes undesired subeffecting like $(\mathtt{put})^{\uparrow} \olessthan \mathbb{0}$
- Condition (2): if $(\mathsf{op})^{\uparrow} \olessthan \varepsilon$ and $(\mathsf{op'})^{\uparrow} \odot \varepsilon' = \varepsilon$ and $\mathsf{op} \neq \mathsf{op'}$, then $(\mathsf{op})^{\uparrow} \olessthan \varepsilon'$
  - Intuition: a handler for op' can only remove $(\mathsf{op'})^{\uparrow}$ from an effect
  - This condition excludes undesired removing like $(\mathtt{put})^{\uparrow} \odot \mathbb{0} = (\mathtt{get})^{\uparrow}$

Our Contribution:
Main Theorem: Soundness

Theorem (Soundness):
any instance of λ_{EA} is sound if it meets the safety conditions

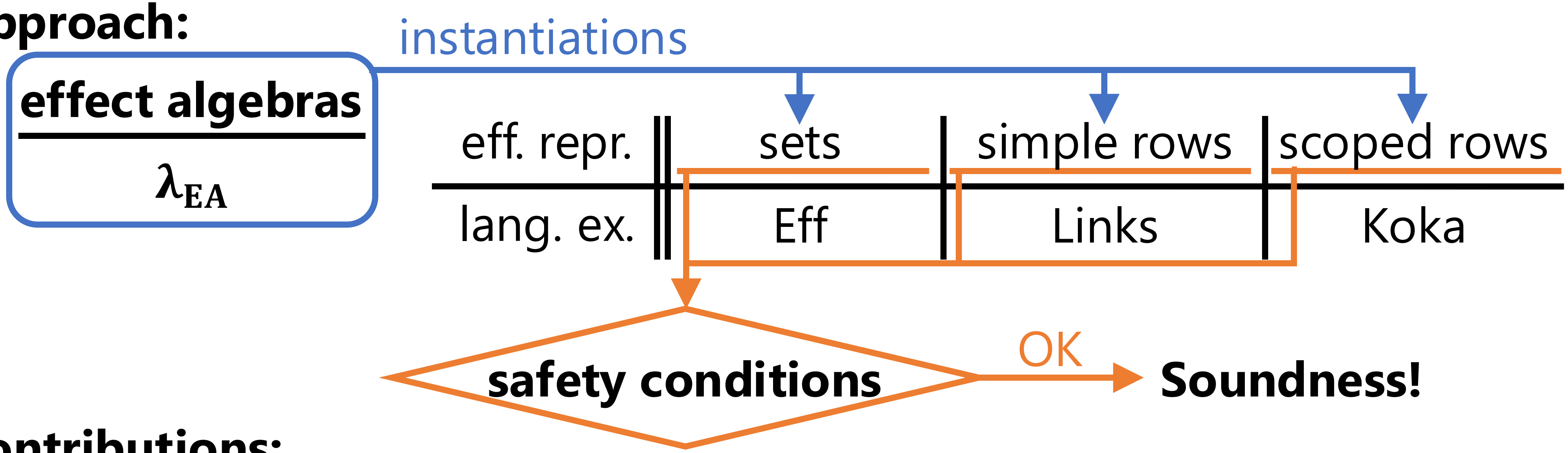
- The safety conditions formalize **a common nature** of sound effect systems
- All we need to do when considering new style of effect representations is to check that an instance meets the safety conditions

Corollary: instances of λ_{EA} for Eff, Links, and Koka, are sound

- Proof: their effect representations meet the safety conditions
- These instances are adaptations of the existing languages to our setting

Summary

Approach:



Contributions:

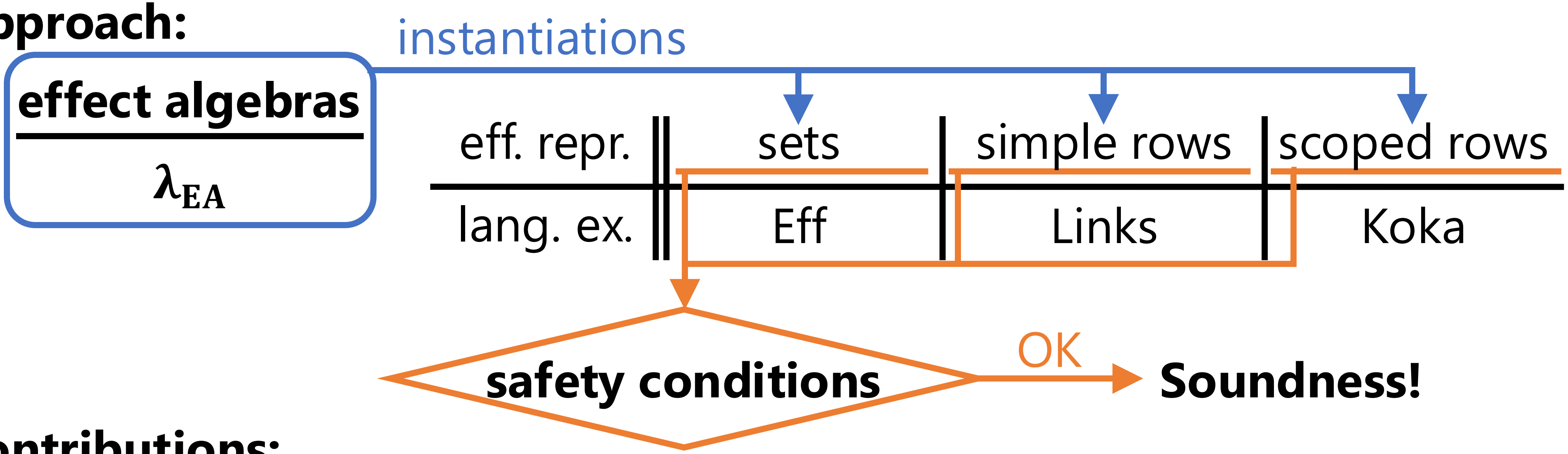
- **Effect algebras:** abstraction of effect representations and their manipulations
- $\lambda_{\mathbf{EA}}$: a λ-calculus with effect handlers, parameterized over effect algebras
- **Safety conditions:** requirements on effect algebras for soundness
- **Main Theorem:** any instance of λ_{EA} is sound if it meets safety conditions

Other Contributions

- We adapt λ_{EA} to **shallow handlers** [Kammar+ 2013]
- We also make **two language extensions** to λ_{EA} and give **additional safety conditions** for them
  - Extension (1): lift coercions [Biernacki+ 2018]
    - Lift coercions are introduced to prevent unintended handlings
  - Extension (2): type-erasure semantics [Biernacki+ 2019]
    - Type-erasure semantics would be a way to efficiently implement effect handlers
  - Our additional safety conditions for these extensions reveal essential **differences** among various effect representations

Summary

Approach:



Contributions:

- **Effect algebras:** abstraction of effect representations and their manipulations
- $\lambda_{\mathbf{EA}}$: a λ-calculus with effect handlers, parameterized over effect algebras
- **Safety conditions:** requirements on effect algebras for soundness
- **Main Theorem:** any instance of λ_{EA} is sound if it meets safety conditions