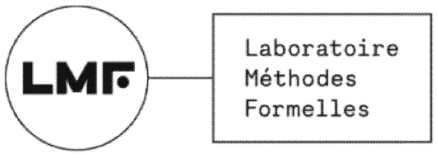
Towards a translation from $\mathbb{K}$ to Dedukti

Amélie LEDEIN, Valentin BLOT et Catherine DUBOIS

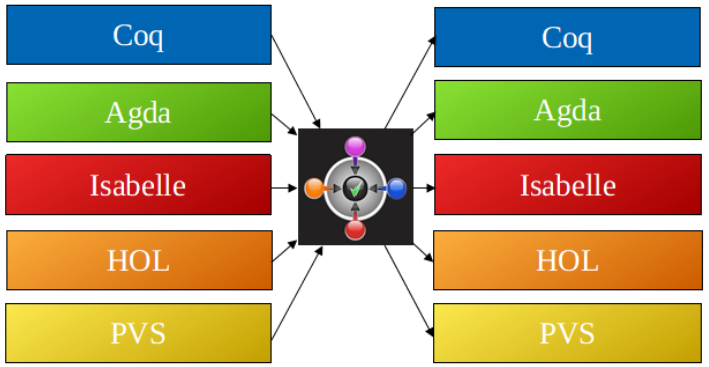


I have a dream...

Allow interoperability of semantics

I have a dream...

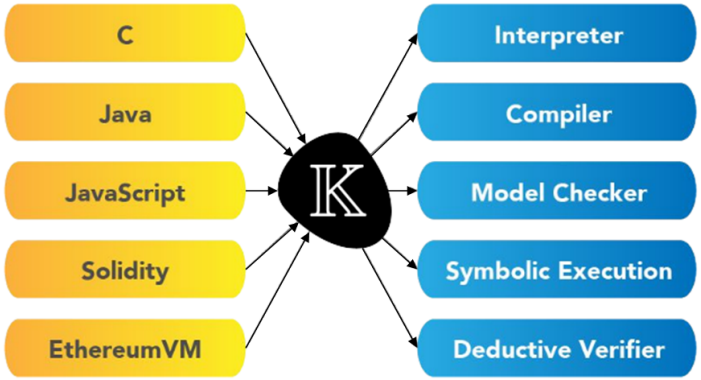
Allow interoperability of semantics

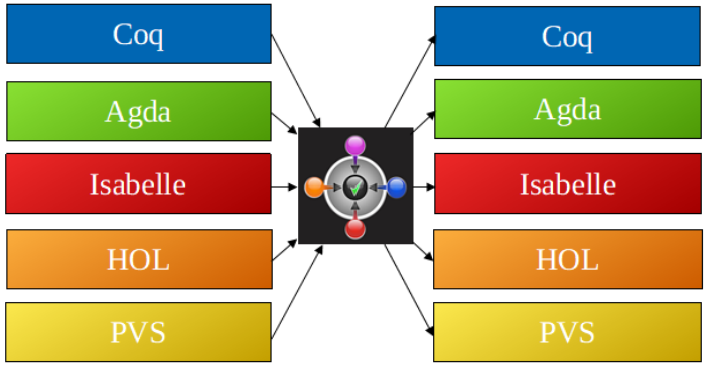


- ✓ Logical framework
- ✓ Various logics encoded in Dedukti
- ✓ Interoperability of proofs

I have a dream...

Allow interoperability of semantics



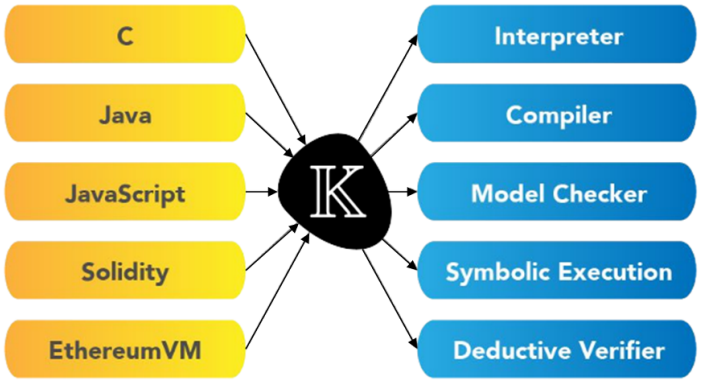


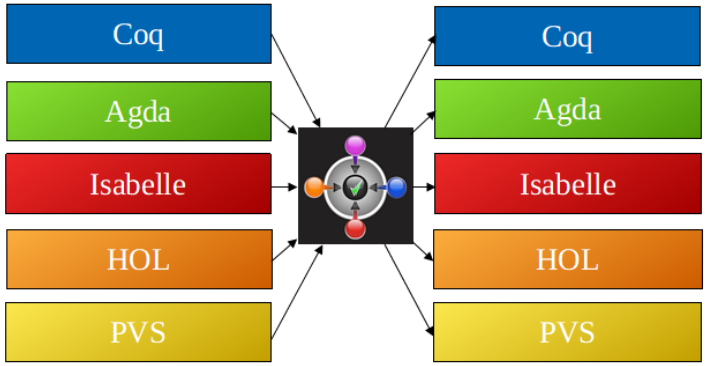
- ✓ Semantical framework
- ✓ Various semantics written in $\mathbb{K}$
- ✓ Execution of semantics
- ✓ Properties on a program (KProver)

- ✓ Logical framework
- ✓ Various logics encoded in Dedukti
- ✓ Interoperability of proofs

I have a dream...

Allow interoperability of semantics





We are interested here in the translation of a semantics written in $\mathbb{K}$ into Dedukti, while keeping the possibility of executing the semantics.

Overview of ecosystems

$\mathbb{K}$

- Semantical framework
  - to define formal semantics of programming languages
  - to automatically generate tools from these semantics

Dedukti

- Logical framework
  - to encode various logics
  - to allow interoperability of proofs between different formal tools

Overview of ecosystems

$\mathbb{K}$

- Semantical framework
  - to define formal semantics of programming languages
  - to automatically generate tools from these semantics

- Based on Matching Logic
  - $\rightarrow$ an untyped 1st order logic with fixpoints and a "next" operator

Dedukti

- Logical framework
  - to encode various logics
  - to allow interoperability of proofs between different formal tools

- Based on $\lambda\Pi$-calculus modulo theory
  - $\rightarrow$ a λ-calculus with dependent types, and extended with rewriting rules

Overview of ecosystems

$\mathbb{K}$

- Semantical framework
  - to define formal semantics of programming languages
  - to automatically generate tools from these semantics
- Based on Matching Logic
  - $\rightarrow$ an untyped 1st order logic with fixpoints and a "next" operator

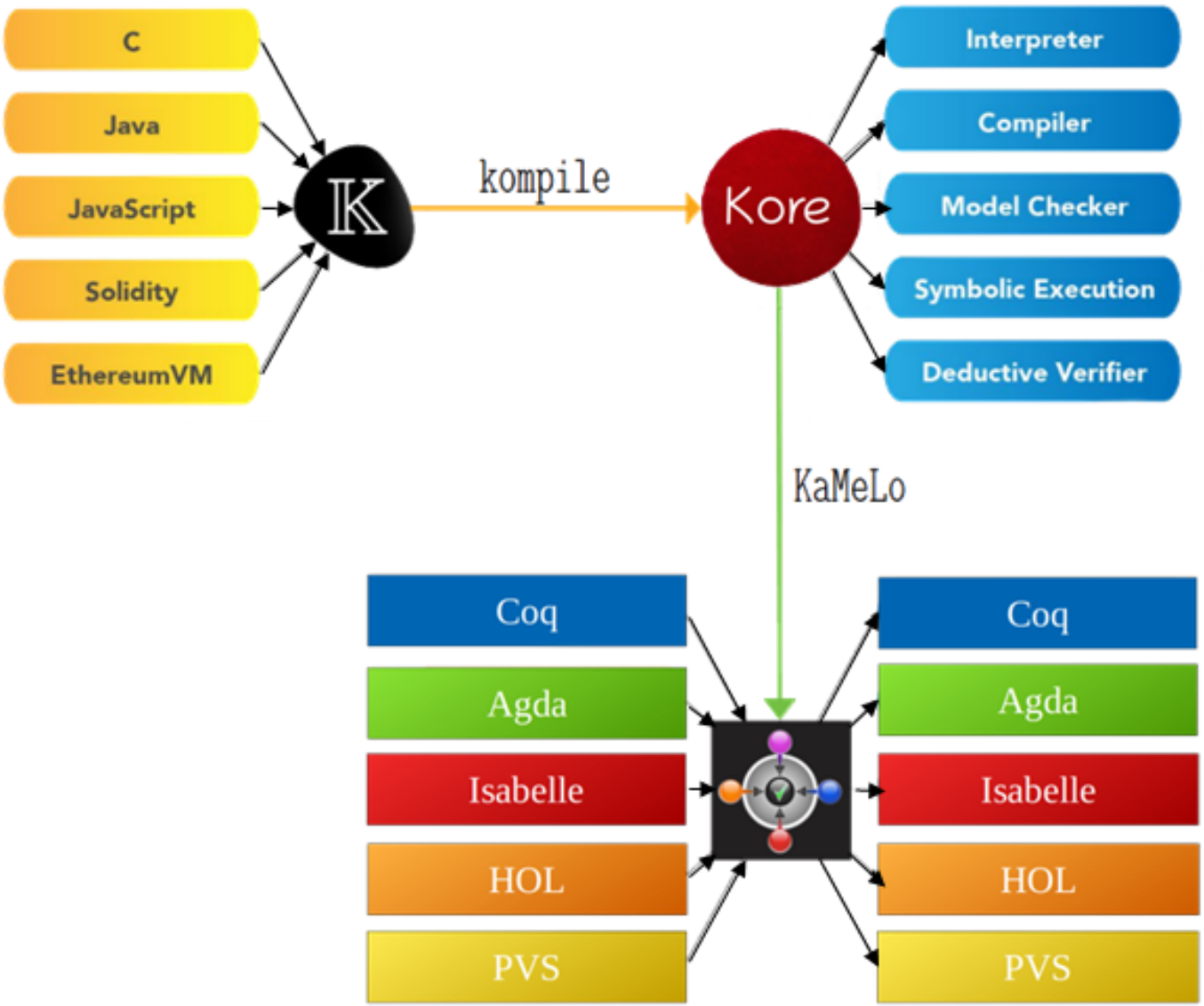
Dedukti

- Logical framework
  - to encode various logics
  - to allow interoperability of proofs between different formal tools
- Based on $\lambda\Pi$-calculus modulo theory
  - $\rightarrow$ a λ-calculus with dependent types, and extended with rewriting rules

- Common feature: $\mathbb{K}$ and Dedukti are based on rewriting.

Characteristic of rewriting	$\mathbb{K}$	Dedukti
At any position	✓	✓
Non-linearity	✓	✓
Conditional	✓	✗
Rewriting modulo ACUI	✓	✗

KaMeLo in action



1. Define semantics in $\mathbb{K}$
2. Translate $\mathbb{K}$ semantics into Dedukti
3. Conclusion

Overview of $\mathbb{K}$

Two steps to define a $\mathbb{K}$ semantics:

- **Syntax**
- **Semantics**

Overview of $\mathbb{K}$

Two steps to define a $\mathbb{K}$ semantics:

- **Syntax**
  - **BNF grammar**
- **Semantics**

Overview of $\mathbb{K}$

Two steps to define a $\mathbb{K}$ semantics:

- **Syntax**
  - **BNF grammar**
- **Semantics**
  - **Configuration** = State of the program
    Example: $\langle\langle\ x + 17\ \rangle_k\ \langle\ x \mapsto 25\ \rangle_{env}\rangle$
  - **Rewriting rule** on configurations ($\sim$ transition system)

Overview of $\mathbb{K}$

$\langle$ `x = 1 ; while 0 < x { x-- } ;` $\rangle_k$
$\langle$ `nil` $\rangle_{env}$

$\langle$ `while 0 < x { x-- } ;` $\rangle_k$
$\langle$ `x` $\mapsto$ `1` $\rangle_{env}$

$\langle$ `while 0 < x { x-- } ;` $\rangle_k$
$\langle$ `x` $\mapsto$ `42` $\rangle_{env}$

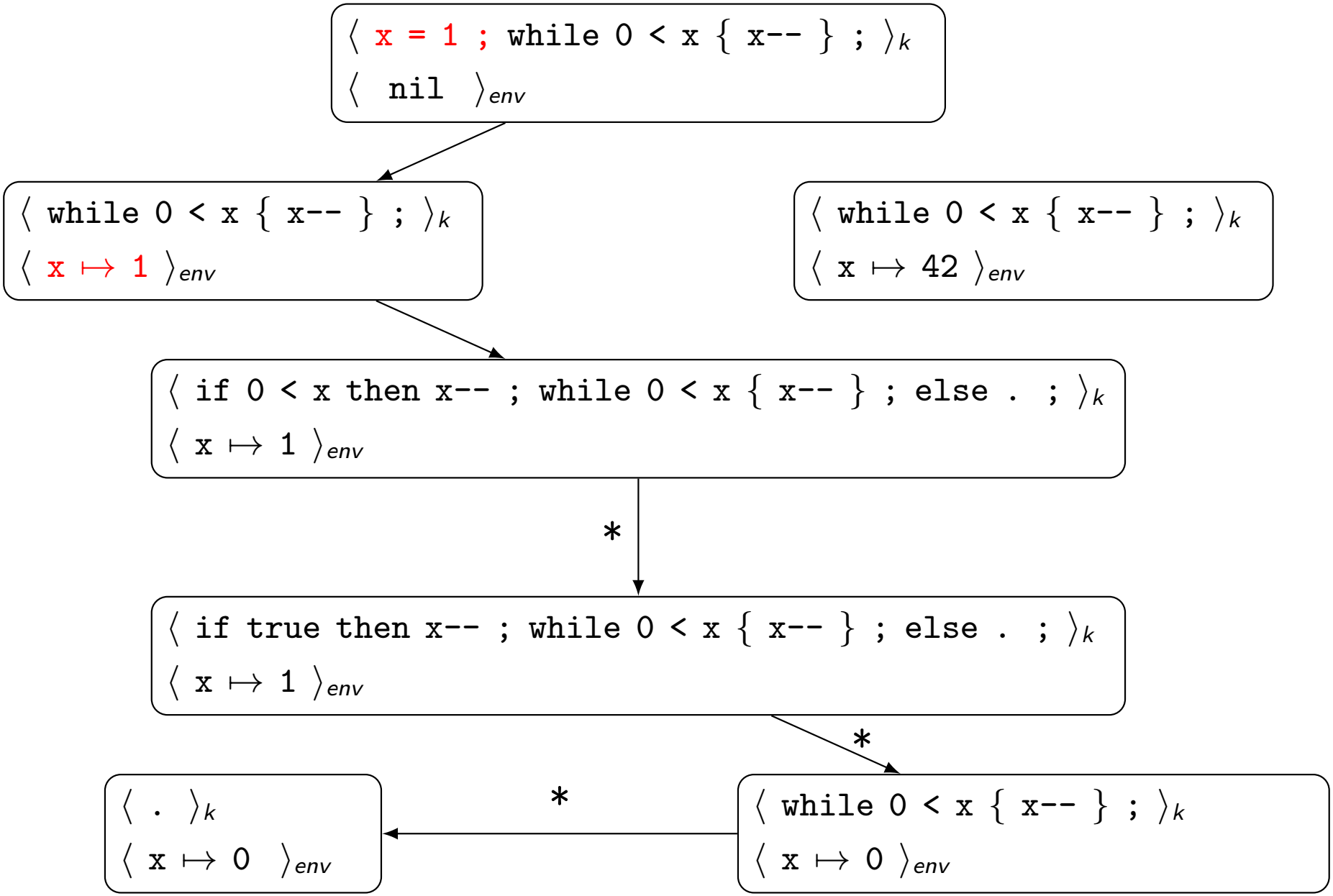
$\langle$ `if 0 < x then x-- ; while 0 < x { x-- } ; else . ;` $\rangle_k$
$\langle$ `x` $\mapsto$ `1` $\rangle_{env}$

$\langle$ `if true then x-- ; while 0 < x { x-- } ; else . ;` $\rangle_k$
$\langle$ `x` $\mapsto$ `1` $\rangle_{env}$

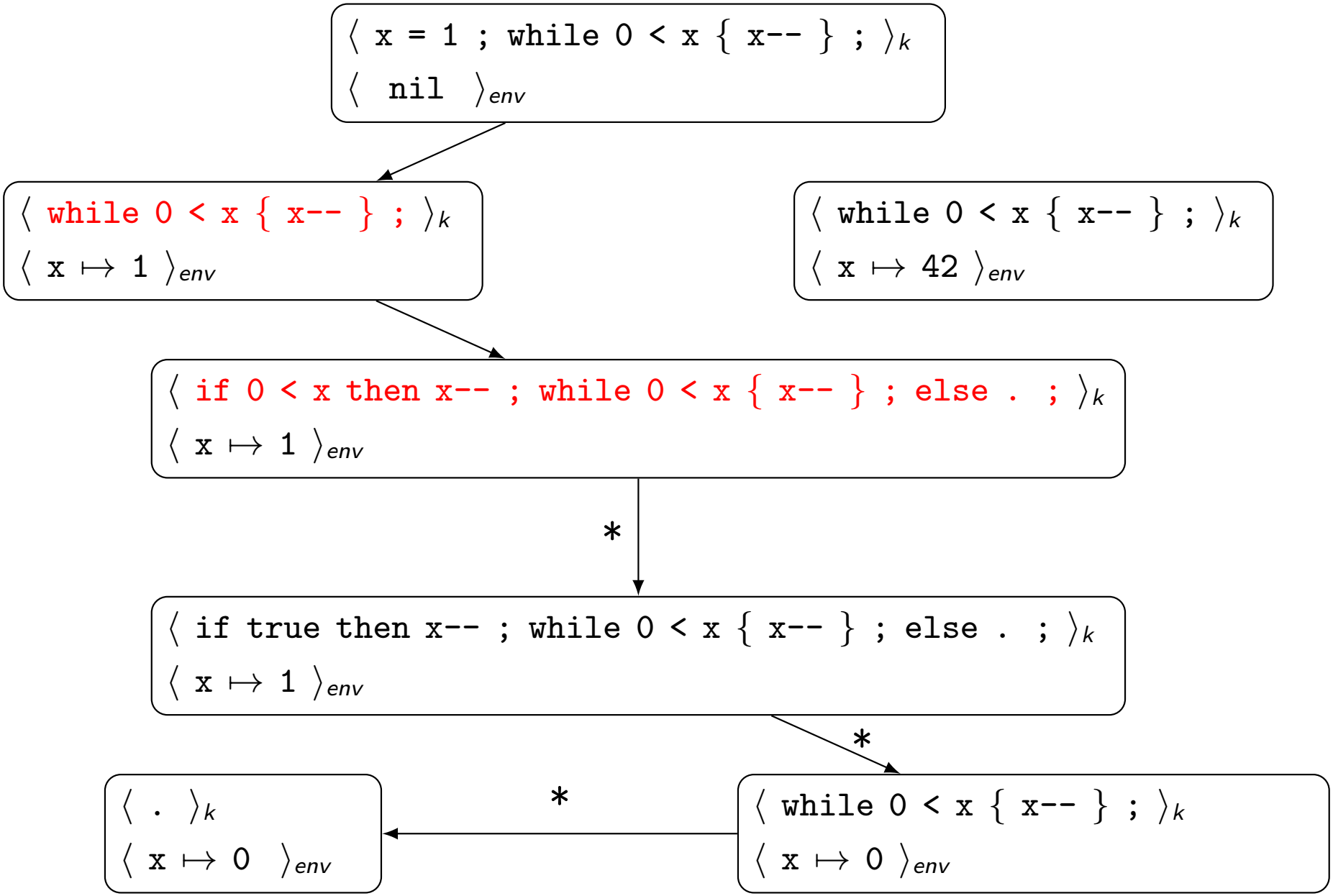
$\langle$ `.` $\rangle_k$
$\langle$ `x` $\mapsto$ `0` $\rangle_{env}$

$\langle$ `while 0 < x { x-- } ;` $\rangle_k$
$\langle$ `x` $\mapsto$ `0` $\rangle_{env}$

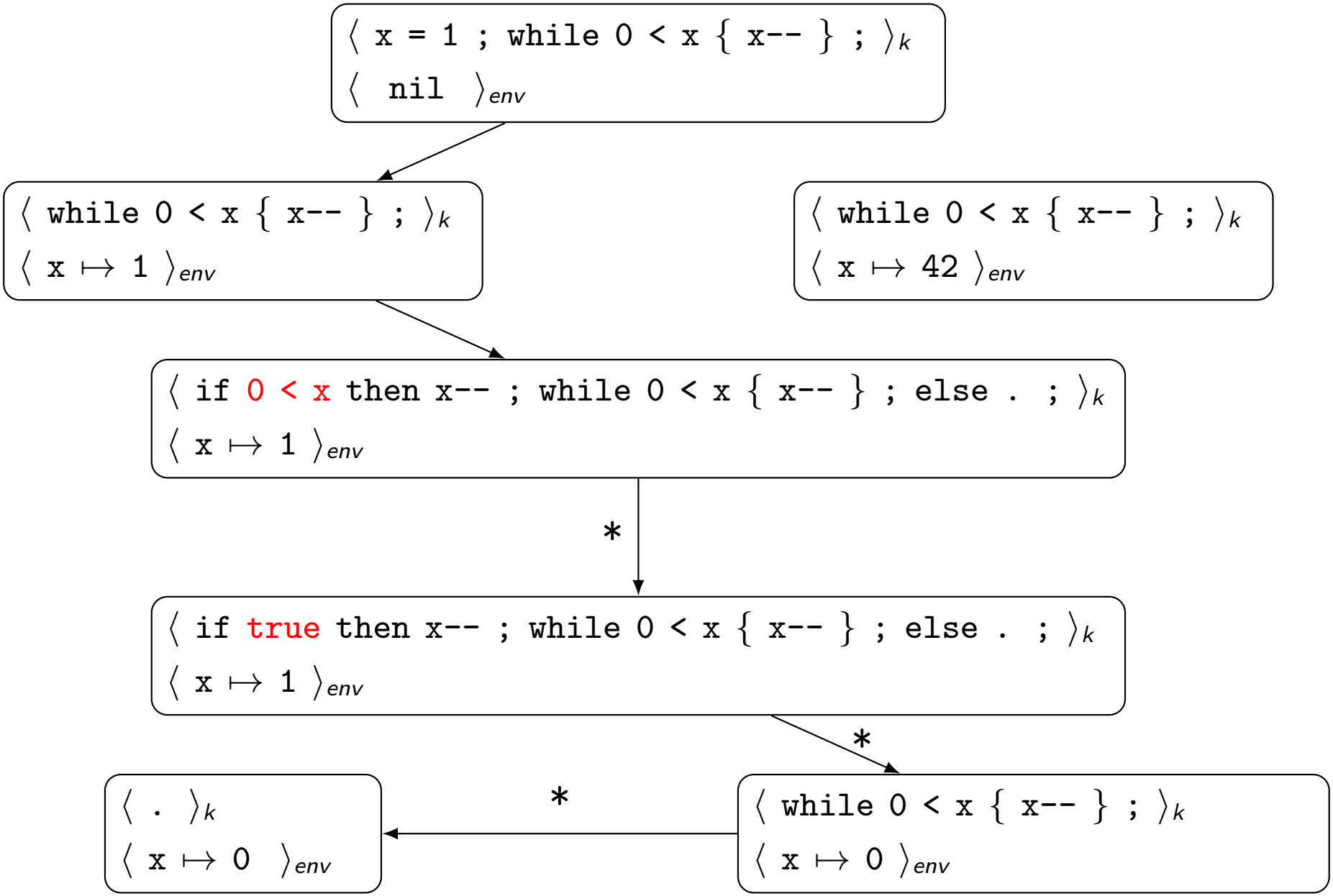
Overview of $\mathbb{K}$



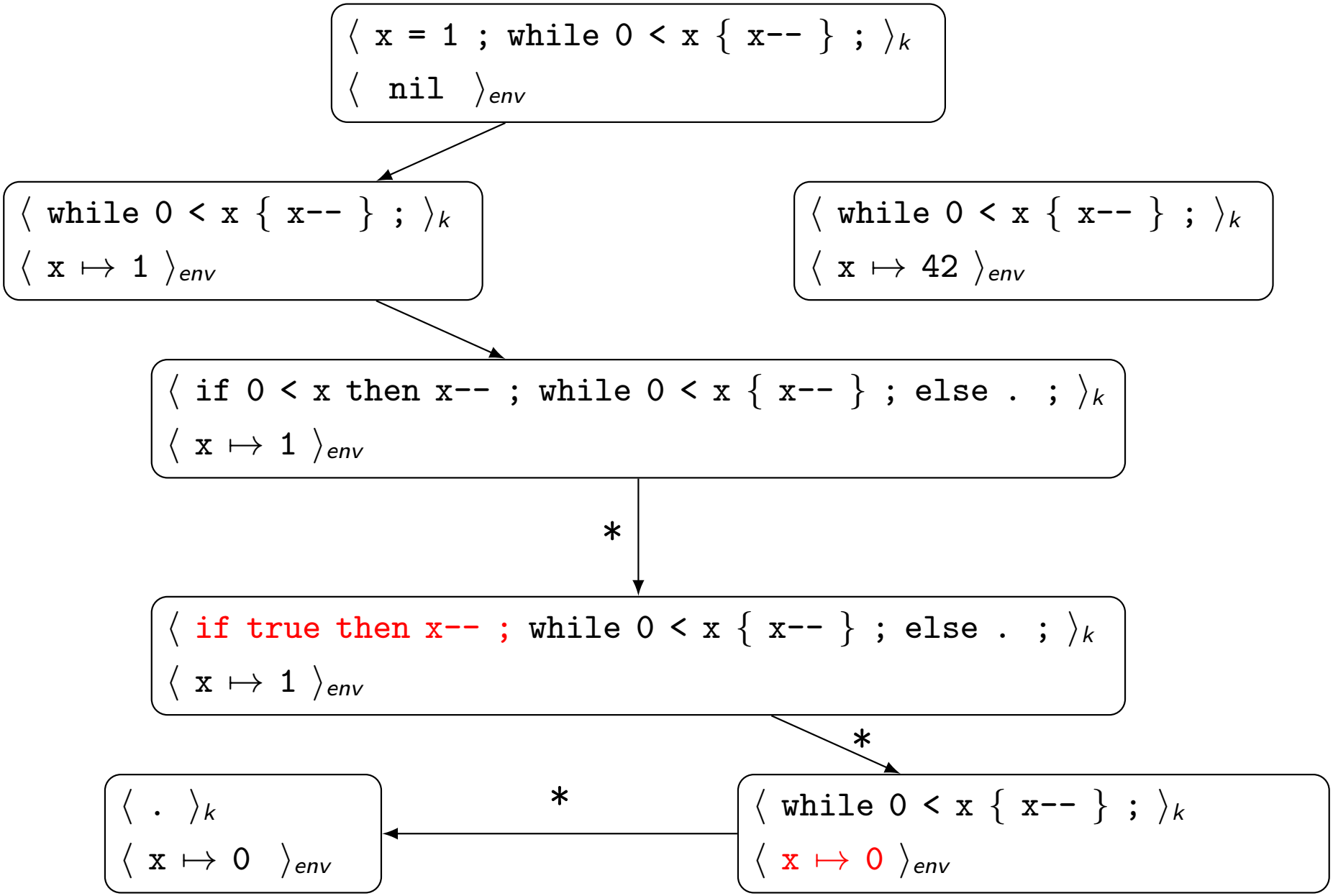
Overview of $\mathbb{K}$



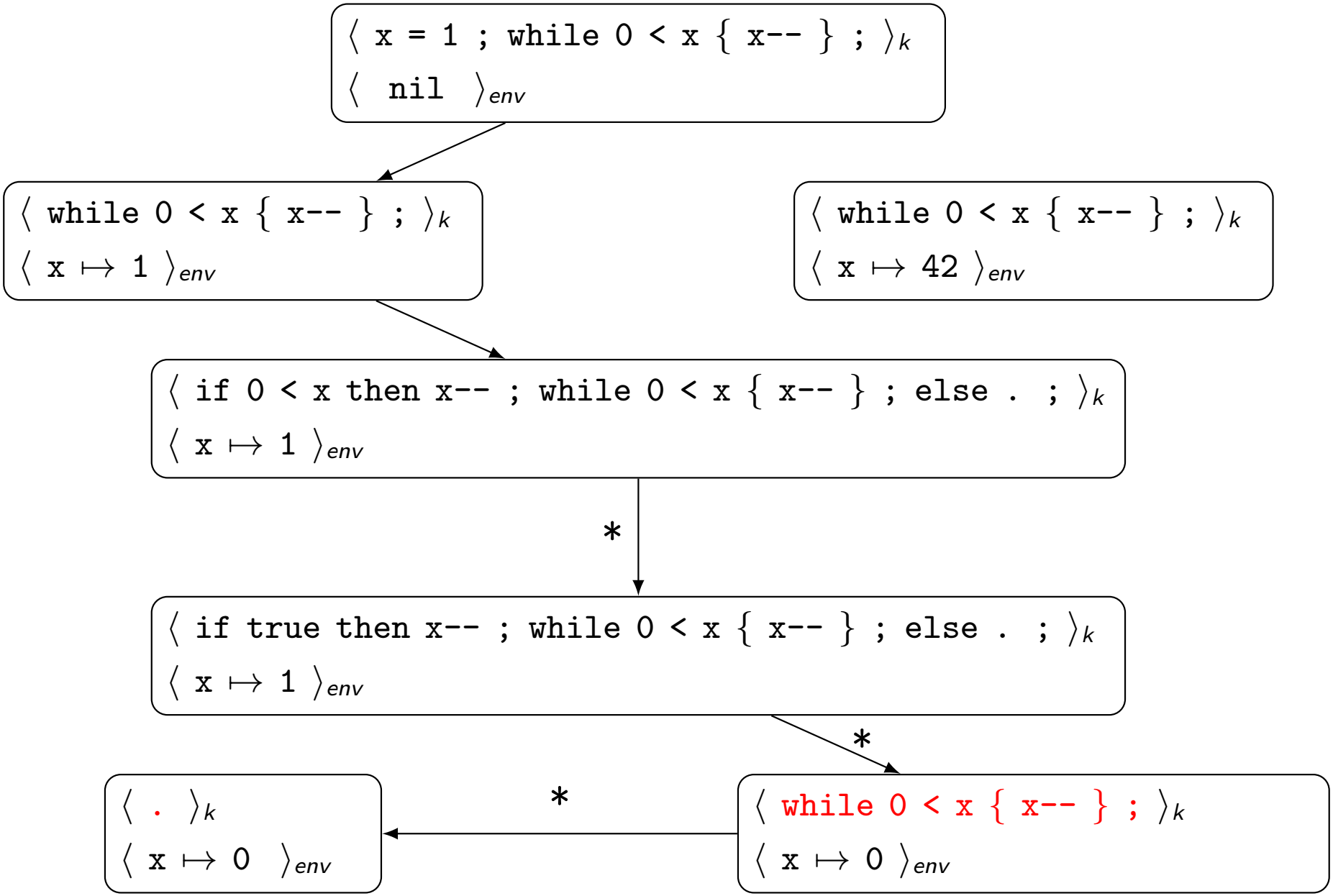
Overview of $\mathbb{K}$



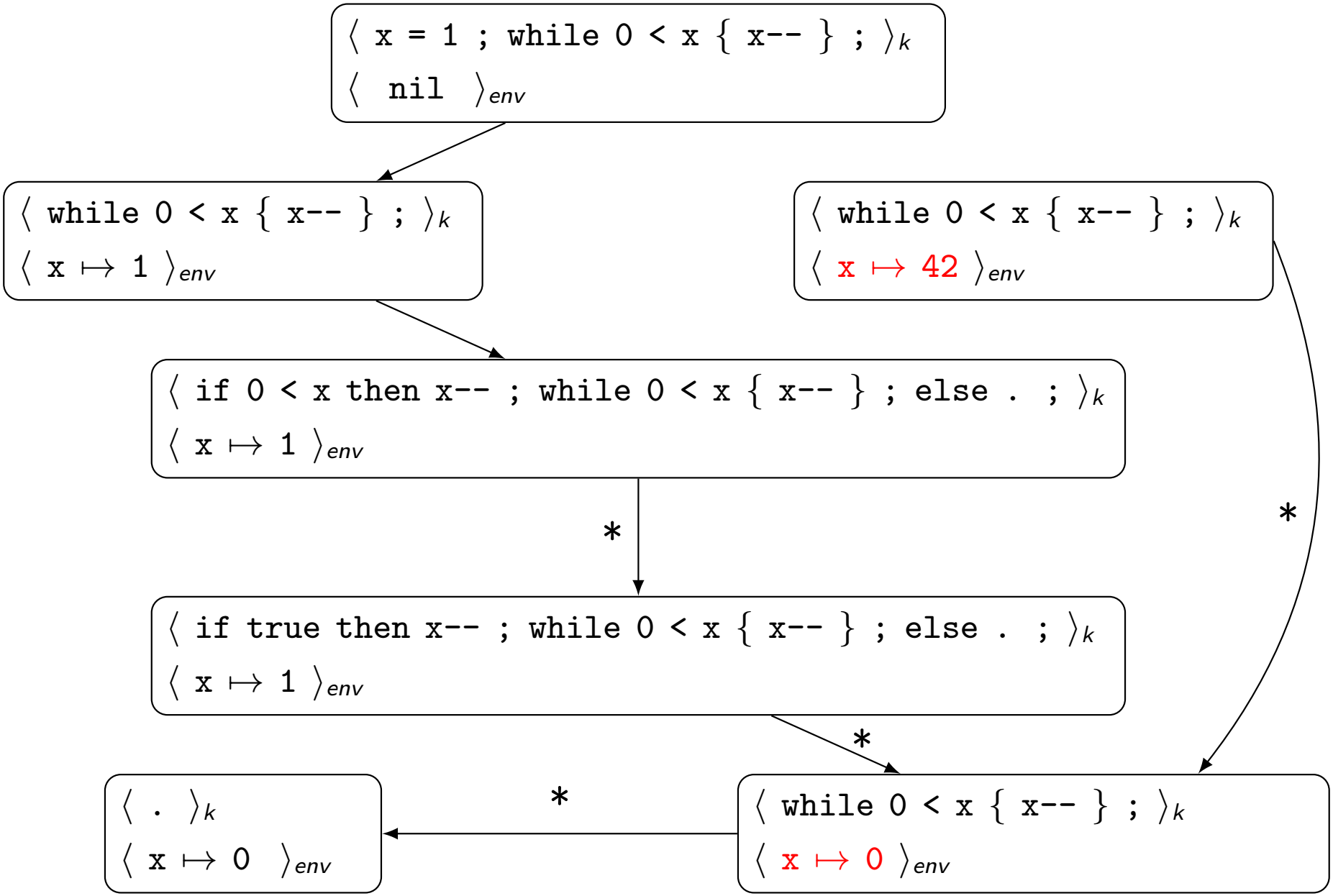
Overview of $\mathbb{K}$



Overview of $\mathbb{K}$

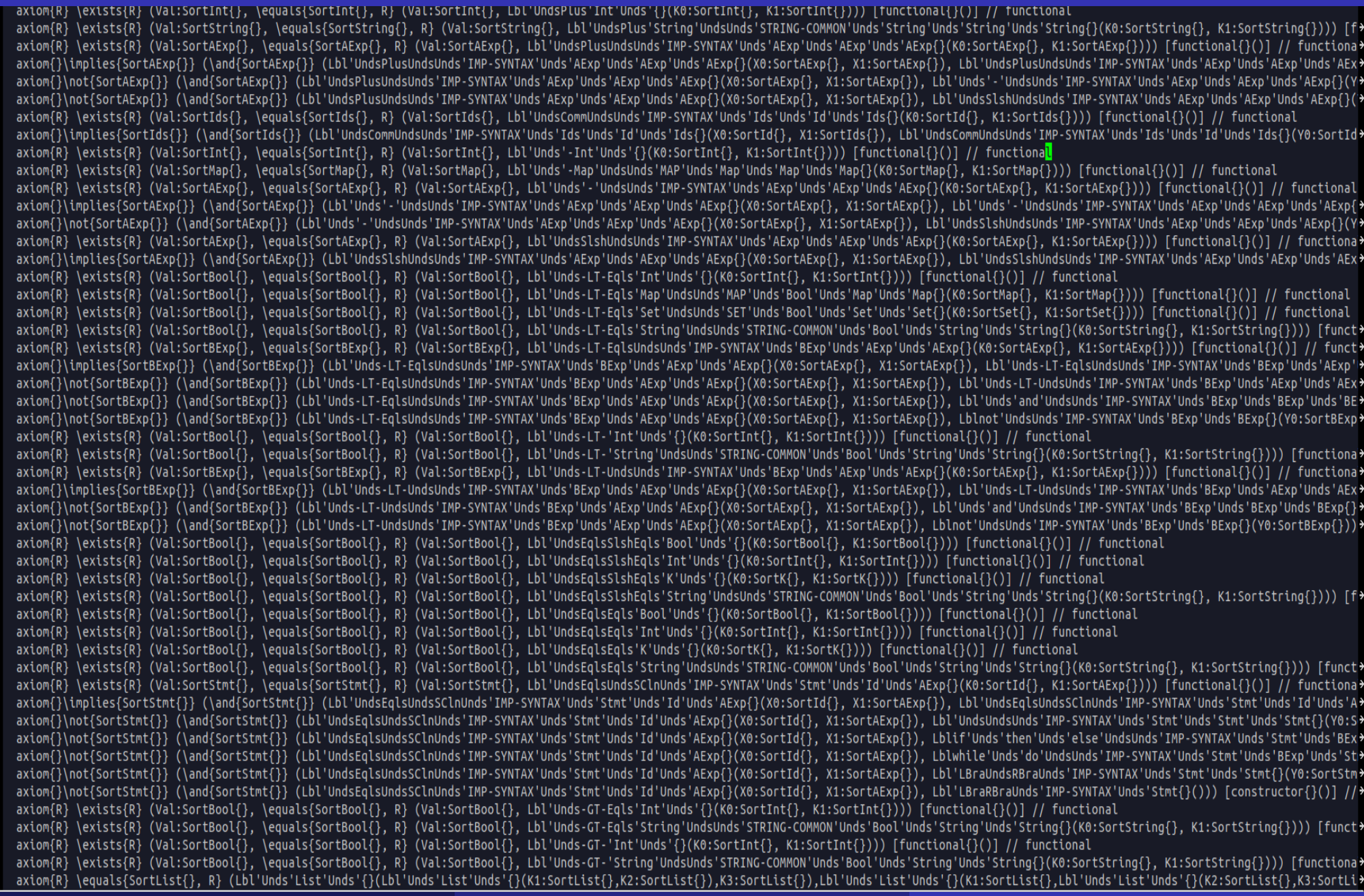


Overview of $\mathbb{K}$



1. Define semantics in $\mathbb{K}$
2. Translate $\mathbb{K}$ semantics into Dedukti
3. Conclusion

Kore: A Matching Logic theory

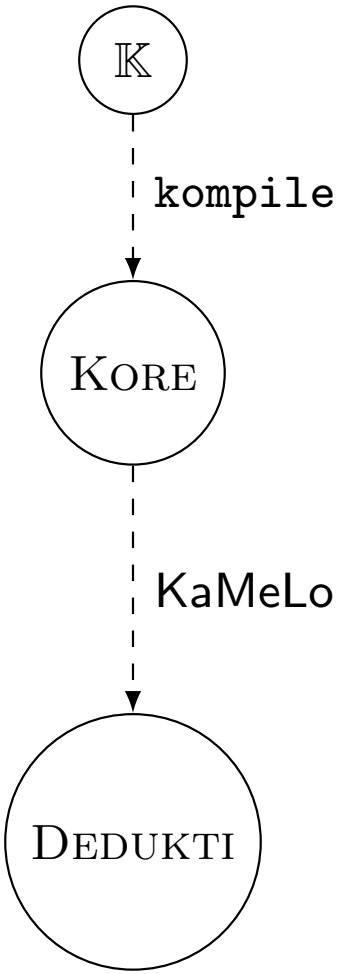


Kore: A Matching Logic theory

A semantics which has 16 lines.

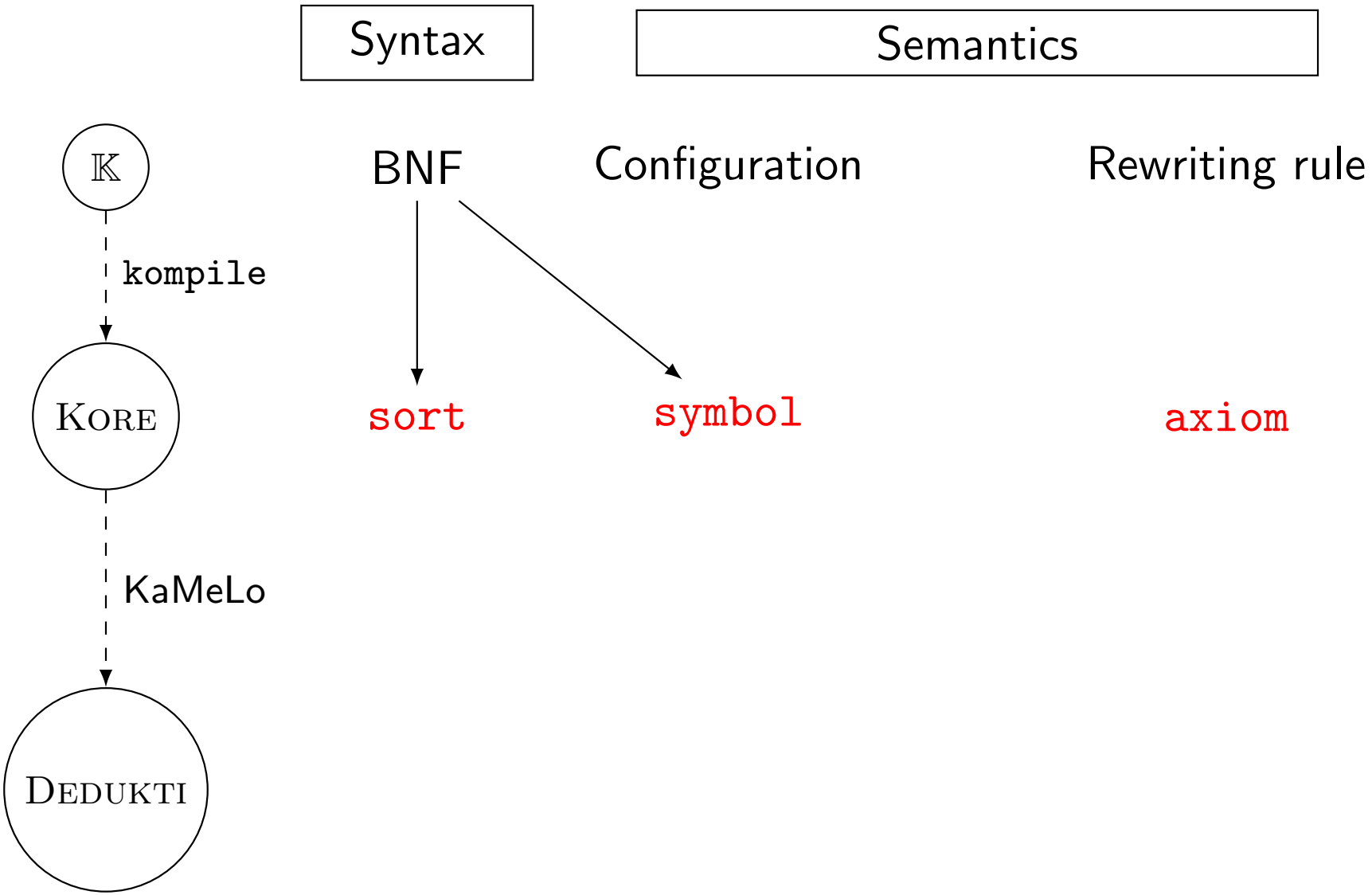
$\sim$A generated Kore file which has about 3,300 lines!

Translation from $\mathbb{K}$ to Dedukti

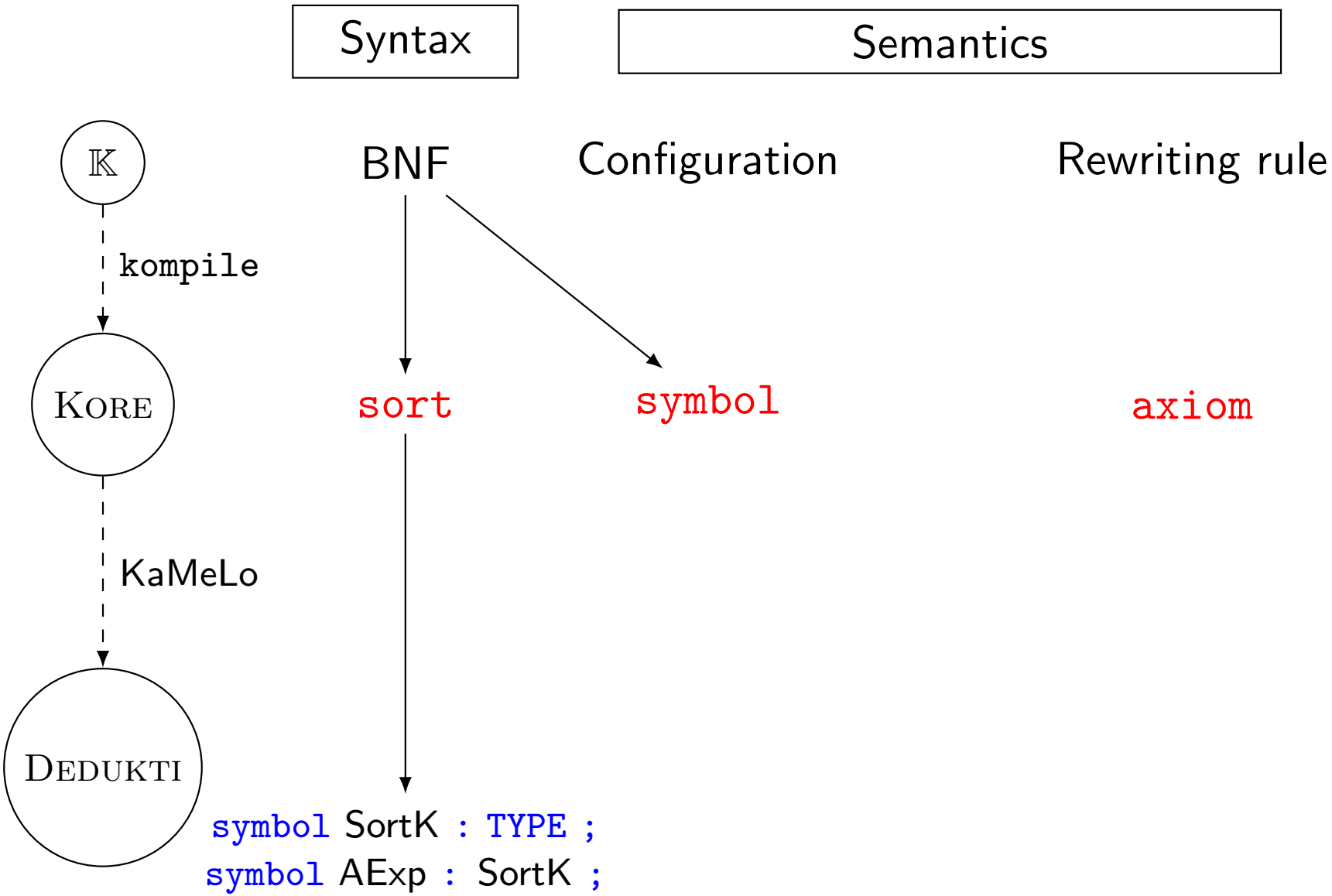


Syntax	Semantics	
BNF	Configuration	Rewriting rule
`sort`	`symbol`	`axiom`

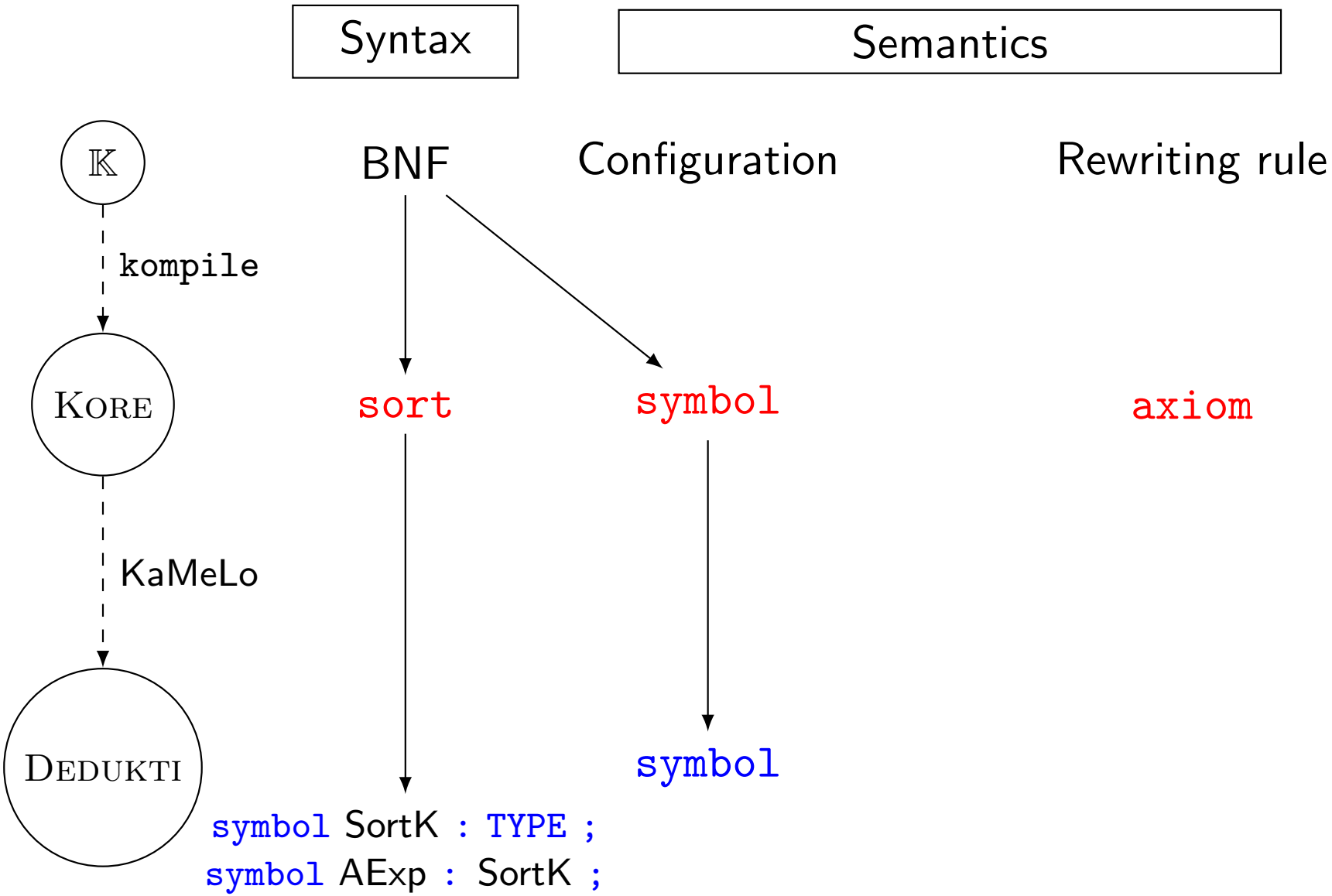
Translation from $\mathbb{K}$ to Dedukti



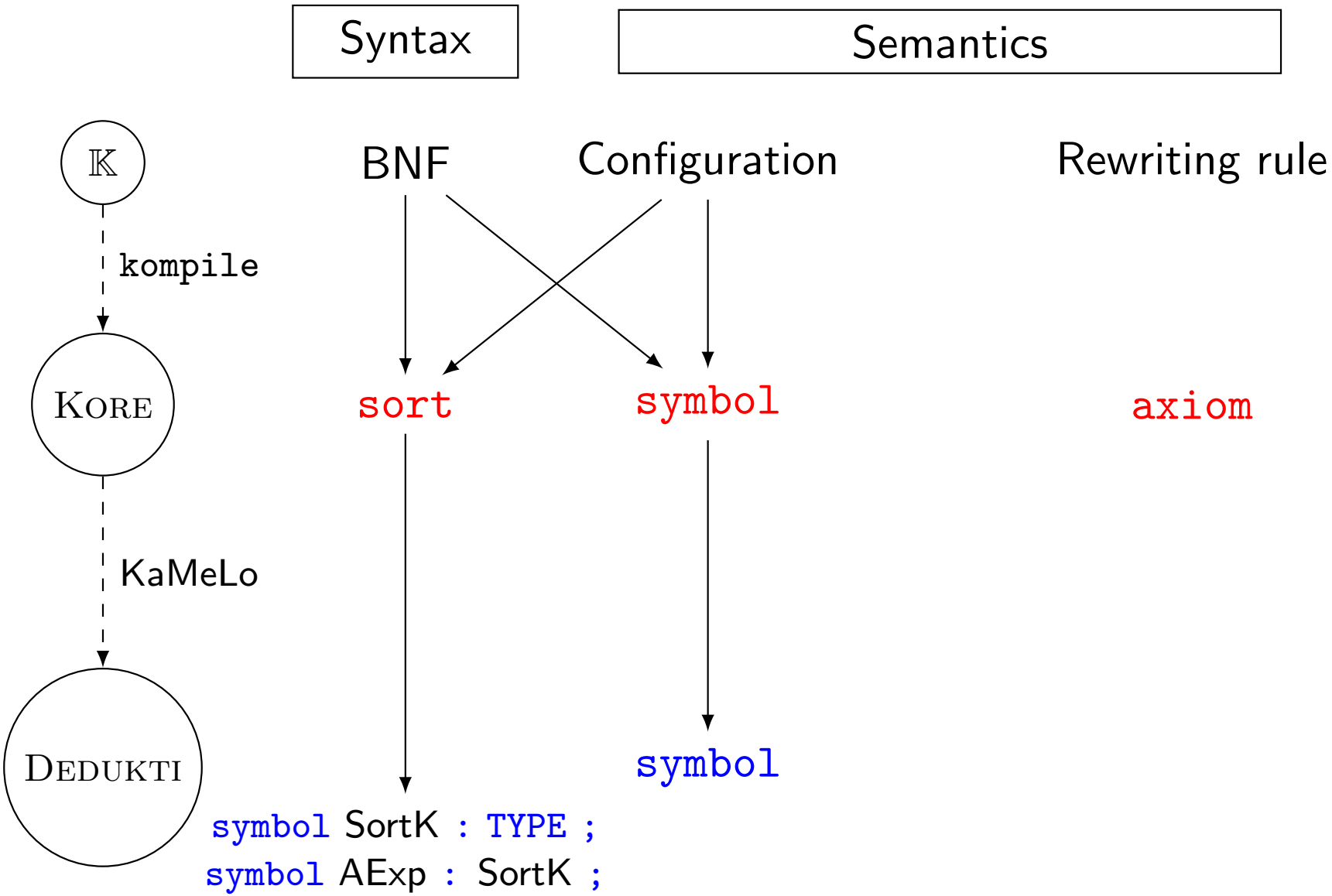
Translation from $\mathbb{K}$ to Dedukti



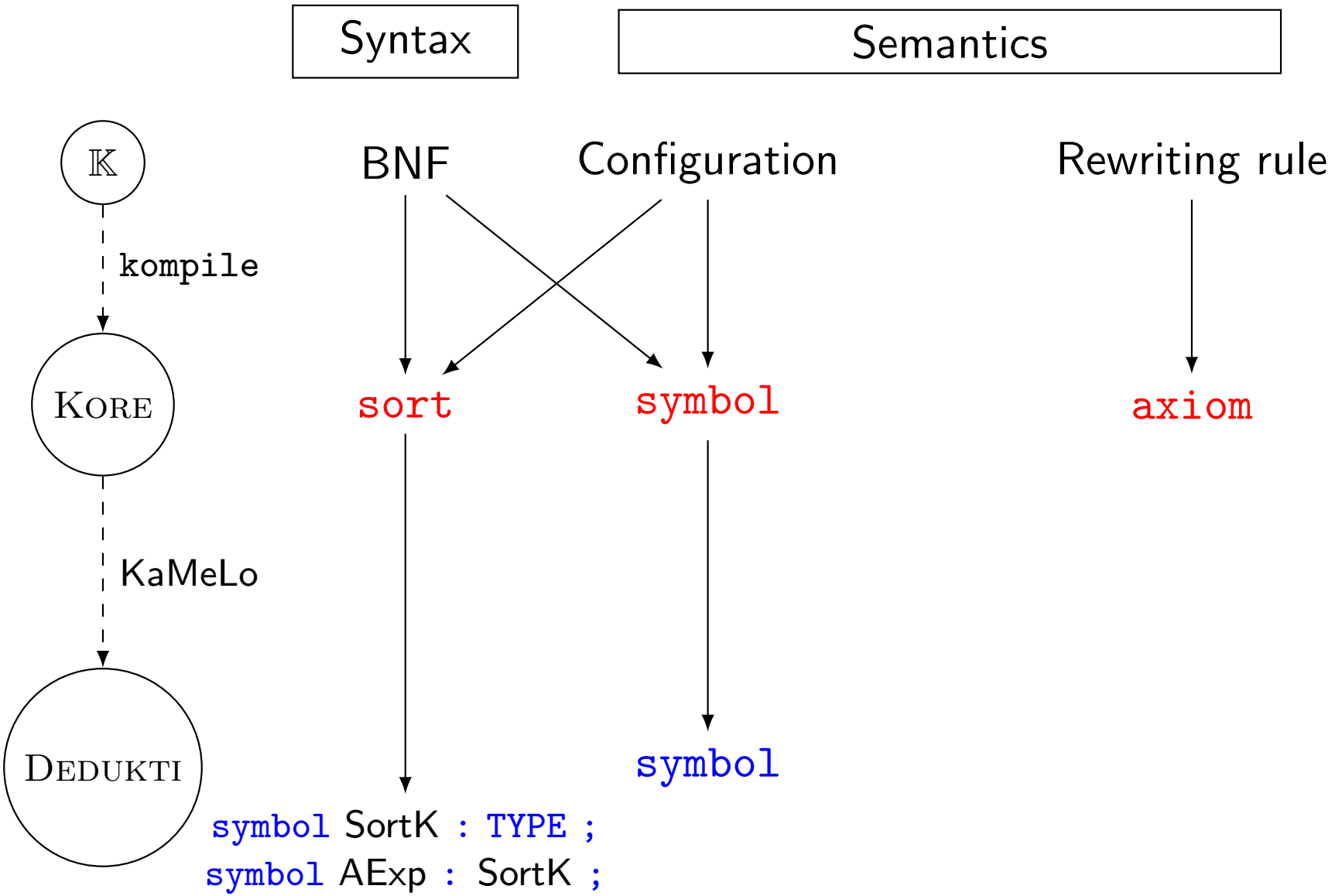
Translation from $\mathbb{K}$ to Dedukti



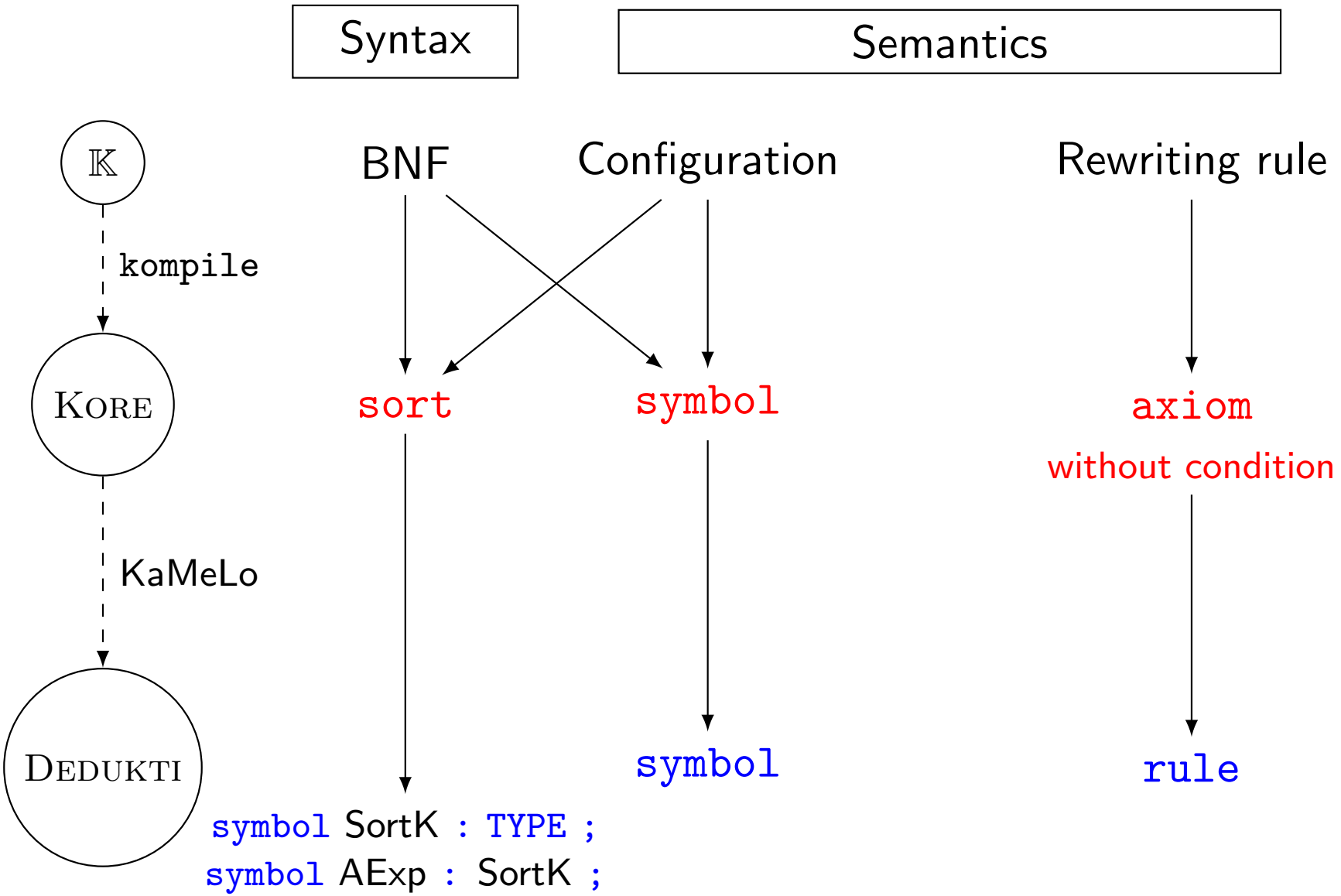
Translation from $\mathbb{K}$ to Dedukti



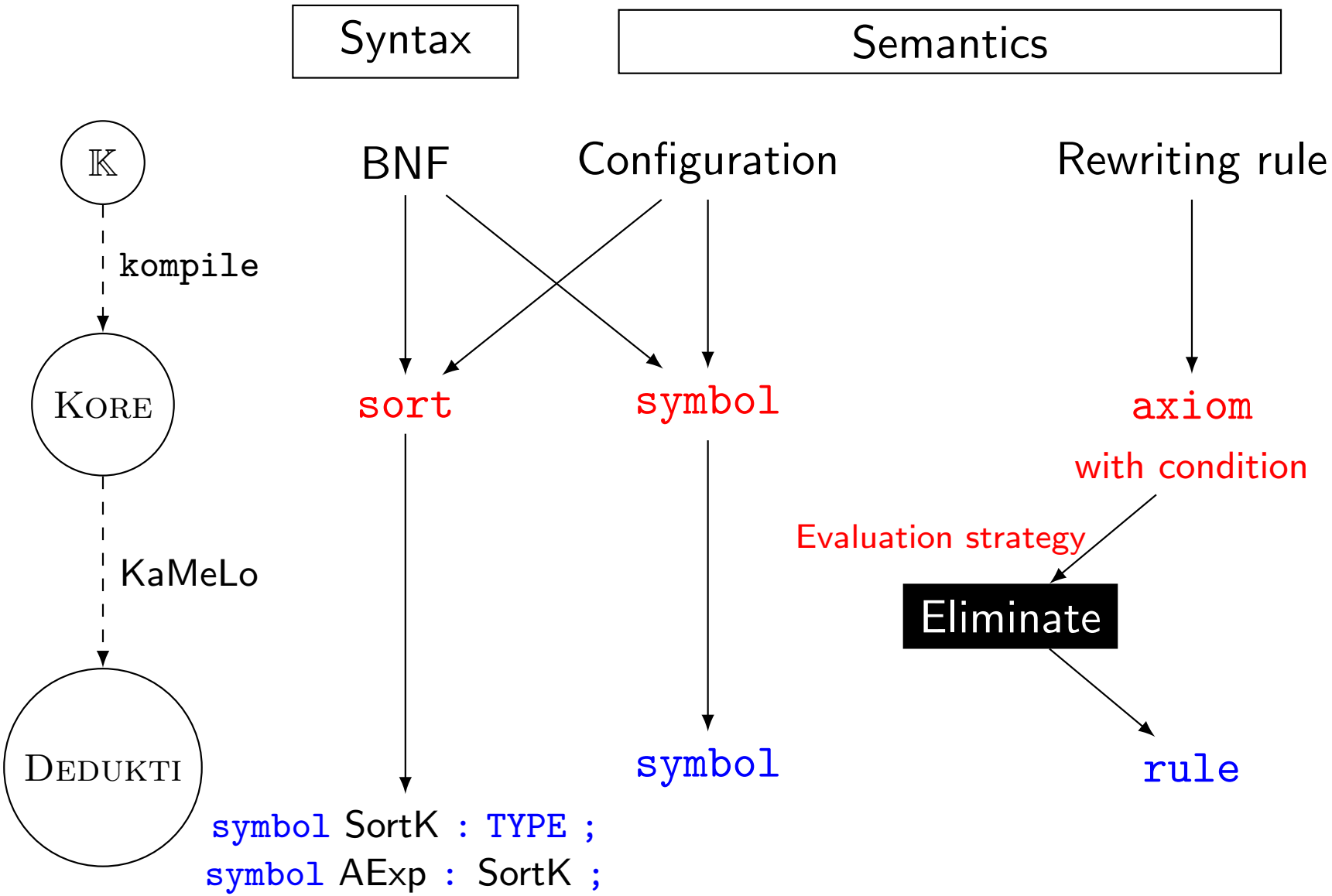
Translation from $\mathbb{K}$ to Dedukti



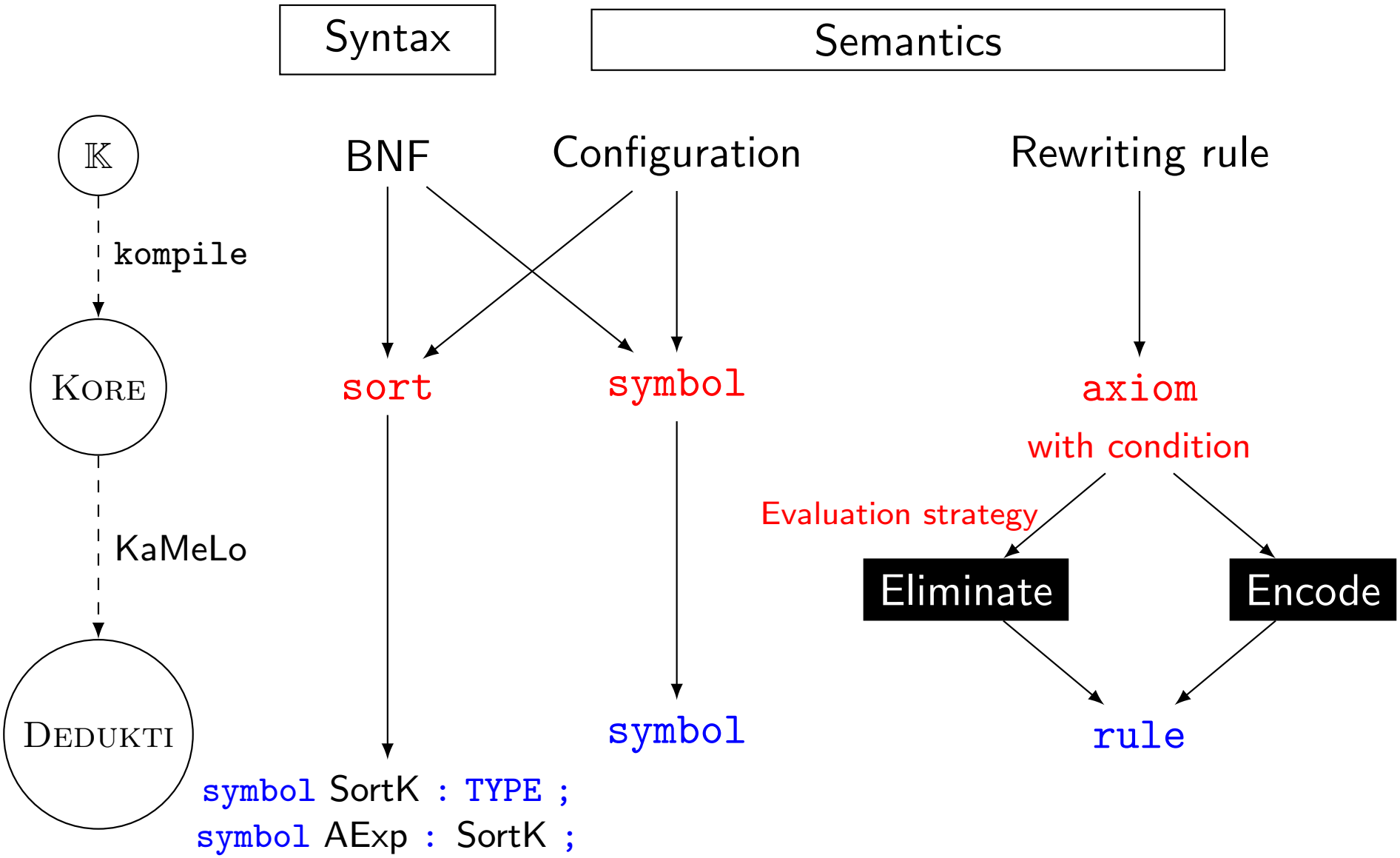
Translation from $\mathbb{K}$ to Dedukti



Translation from $\mathbb{K}$ to Dedukti



Translation from $\mathbb{K}$ to Dedukti



Translate evaluation strategies

Generated rules to define evaluation strategies:

Key ideas:

- **Evaluation is ordering thanks to a list.**
  $(E_1$ `and` $E_2) \curvearrowright E_3 \curvearrowright$.
- **Evaluated expressions have a specific type.**
  `true` `and` `false` has the type **BExp**
  `true` has the type `Bool`

Translate evaluation strategies

Generated rules to define evaluation strategies:

1. `rule` E_1 and E_2 `=>` $E_1 \curvearrowright (\clubsuit^1_{\text{and}}\ E_2)$ `requires` $E_1 \notin$ `Bool`
2. `rule` $E_1 \curvearrowright (\clubsuit^1_{\text{and}}\ E_2)$ `=>` E_1 and E_2 `requires` $E_1 \in$ `Bool`

Translation into DEDUKTI:

1. Instantiation of E_1:
   a. `rule` $\langle\ (\mathit{not}\ \$\texttt{X1})\ \mathit{and}\ \$\texttt{E2} \curvearrowright \$\texttt{s}\ \rangle_k$
      $\hookrightarrow$ $\langle\ (\mathit{not}\ \$\texttt{X1}) \curvearrowright (\clubsuit^1_{\text{and}}\ \$\texttt{E2}) \curvearrowright \$\texttt{s}\ \rangle_k$

   b. `rule` $\langle\ (\$\texttt{X1}\ \mathit{and}\ \$\texttt{X2})\ \mathit{and}\ \$\texttt{E2} \curvearrowright \$\texttt{s}\ \rangle_k$
      $\hookrightarrow$ $\langle\ (\$\texttt{X1}\ \mathit{and}\ \$\texttt{X2}) \curvearrowright (\clubsuit^1_{\text{and}}\ \$\texttt{E2}) \curvearrowright \$\texttt{s}\ \rangle_k$

2. `rule` $\langle\ (\mathit{inj}\ \$\texttt{E1}) \curvearrowright (\clubsuit^1_{\text{and}}\ \$\texttt{E2}) \curvearrowright \$\texttt{s}\ \rangle_k$
   $\hookrightarrow$ $\langle\ (\mathit{inj}\ \$\texttt{E1})\ \mathit{and}\ \$\texttt{E2} \curvearrowright \$\texttt{s}\ \rangle_k$

The grammar of **BExp**:

```
syntax BExp ::= Bool
    | "not" BExp
    > BExp "and" BExp
    | "(" BExp ")"
```

Translate a CTRS to a TRS[1]

- **Example 1**:

  (1) `rule` *max X Y* `=>` *Y* `requires` *X* `<Int` *Y*

  (2) `rule` *max X Y* `=>` *X* `requires` *X* `>=Int` *Y*

[1] Patrick Viry, *Elimination of Conditions*, Journal of Symbolic Computation, 1999

Translate a CTRS to a TRS[1]

- **Example 1**:

  (1) `rule` *max X Y* `=>` *Y* `requires` *X* `<Int` *Y*

  (2) `rule` *max X Y* `=>` *X* `requires` *X* `>=Int` *Y*

  translated into

  (0) `rule` *max* `$x $y` $\hookrightarrow$ ♭*max* `$x $y` (`$x` $<$ `$y`) (`$x` $\geq$ `$y`)

[1] Patrick Viry, *Elimination of Conditions*, Journal of Symbolic Computation, 1999

Translate a CTRS to a TRS[1]

- **Example 1**:

  (1) `rule` *max X Y* `=>` *Y* `requires` *X* `<Int` *Y*

  (2) `rule` *max X Y* `=>` *X* `requires` *X* `>=Int` *Y*

  translated into

  (0) `rule` *max* `$x $y` $\hookrightarrow$ $\flat$*max* `$x $y` (`$x` $<$ `$y`) (`$x` $\geq$ `$y`)

  (1') `rule` $\flat$*max* `$x $y` *true* `_` $\hookrightarrow$ `$y`

  (2') `rule` $\flat$*max* `$x $y _` *true* $\hookrightarrow$ `$x`

[1] Patrick Viry, *Elimination of Conditions*, Journal of Symbolic Computation, 1999

Translate a CTRS to a TRS[1]

- **Example 1**:

  (1) `rule` *max X Y* `=>` *Y* `requires` *X* `<Int` *Y*
  (2) `rule` *max X Y* `=>` *X* `requires` *X* `>=Int` *Y*

  translated into

  (0) `rule` *max* `$x $y` $\hookrightarrow$ $\flat$*max* `$x $y` (`$x` $<$ `$y`) (`$x` $\geq$ `$y`)
  (1') `rule` $\flat$*max* `$x $y` *true* `_` $\hookrightarrow$ `$y`
  (2') `rule` $\flat$*max* `$x $y _` *true* $\hookrightarrow$ `$x`

- **Example 2**:

  (A) `rule` *max X Y* `=>` *Y* `requires` *X* `<Int` *Y*
  (B) `rule` *max X Y* `=>` *X* `[owise ]`

  translated into

  ($\aleph$) `rule` *max* `$x $y` $\hookrightarrow$ $\flat$*max* `$x $y` (`$x` $<$ `$y`)
  (A') `rule` $\flat$*max* `$x $y` *true* $\hookrightarrow$ `$y`
  (B') `rule` $\flat$*max* `$x $y` *false* $\hookrightarrow$ `$x`

[1] Patrick Viry, *Elimination of Conditions*, Journal of Symbolic Computation, 1999

1. Define semantics in $\mathbb{K}$
2. Translate $\mathbb{K}$ semantics into Dedukti
3. Conclusion

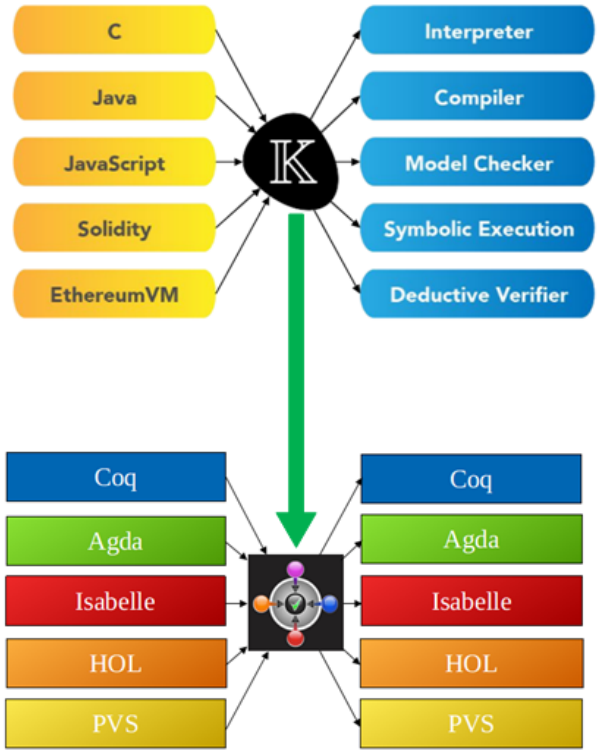
Further work

- **Objectives:**

- **Milestones:**

Further work

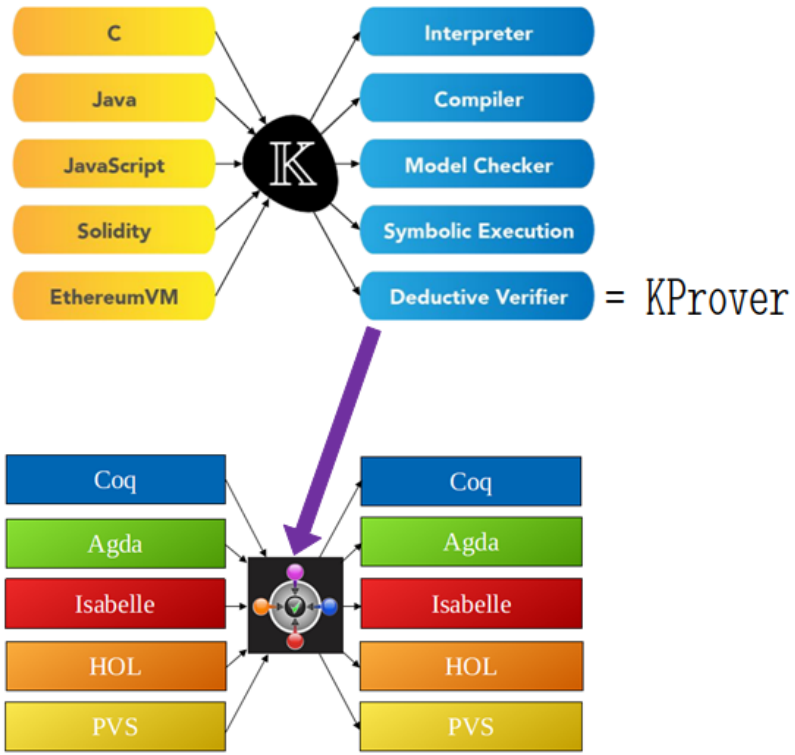
- **Objectives:**
  1. Translate a $\mathbb{K}$ semantics into Dedukti, to execute it.



- **Milestones:**
  1. Test the semantics.

Further work

- **Objectives:**
  1. Translate a $\mathbb{K}$ semantics into Dedukti, to execute it.
  2. Translate a $\mathbb{K}$ semantics into Dedukti, to check proofs.

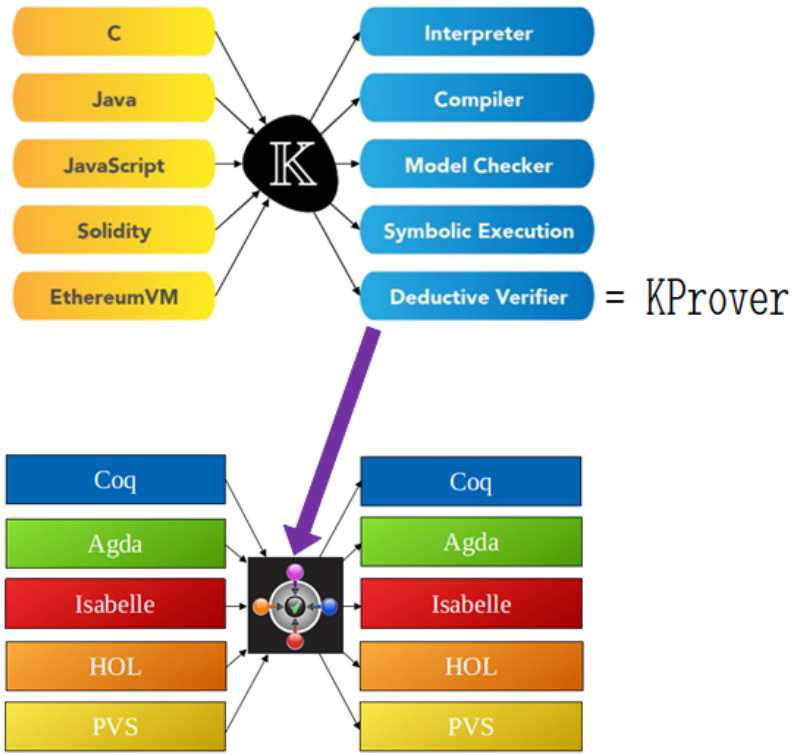


- **Milestones:**
  1. Test the semantics.

Further work

- **Objectives:**
  1. Translate a $\mathbb{K}$ semantics into Dedukti, to execute it.
  2. Translate a $\mathbb{K}$ semantics into Dedukti, to check proofs.

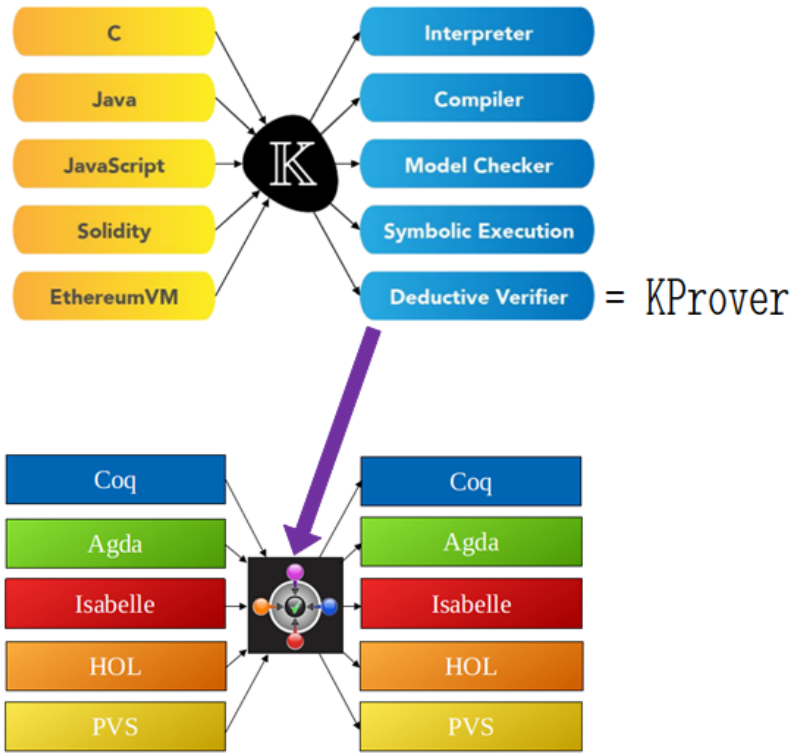
     $\rightarrow$ See my talk at Dedukti school!



- **Milestones:**
  1. Test the semantics.

Further work

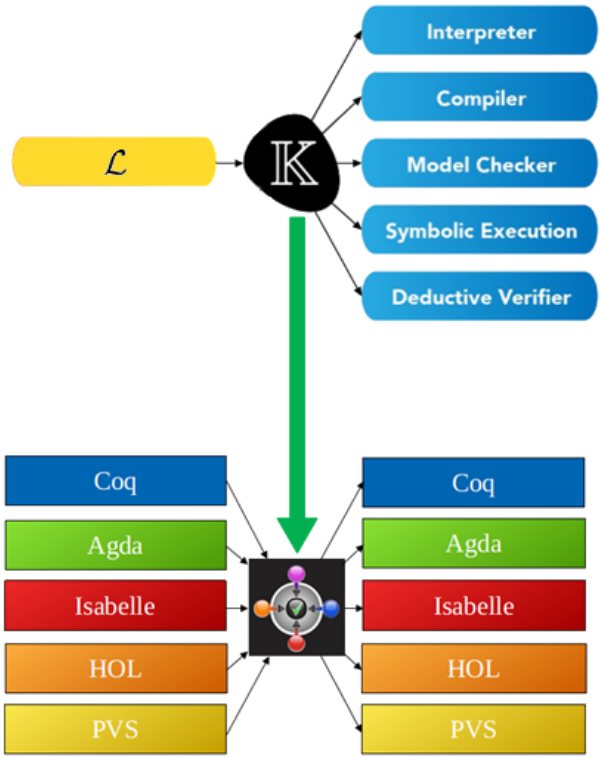
- **Objectives:**
  1. Translate a $\mathbb{K}$ semantics into Dedukti, to execute it.
  2. Translate a $\mathbb{K}$ semantics into Dedukti, to check proofs.
     → See my talk at Dedukti school!



- **Milestones:**
  1. Test the semantics.
  2. Recheck KProver proofs.
     → Work in progress:
     See my talk at WEPN!

Further work

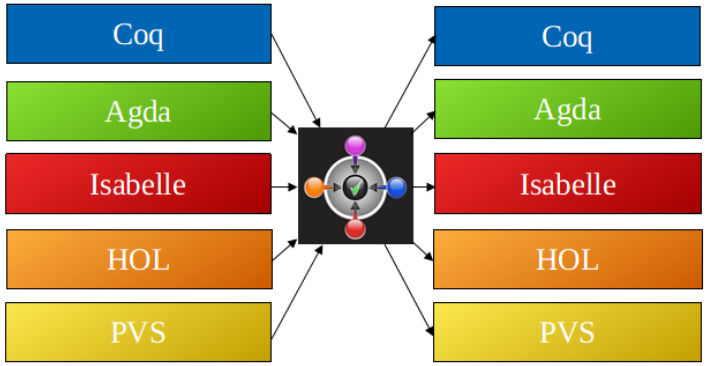
- **Objectives:**
  1. Translate a $\mathbb{K}$ semantics into Dedukti, to execute it.
  2. Translate a $\mathbb{K}$ semantics into Dedukti, to check proofs.
     → See my talk at Dedukti school!



- **Milestones:**
  1. Test the semantics.
  2. Recheck KProver proofs.
     → Work in progress:
     See my talk at WEPN!
  3. Prove properties about a language $\mathcal{L}$ described in $\mathbb{K}$.

Further work

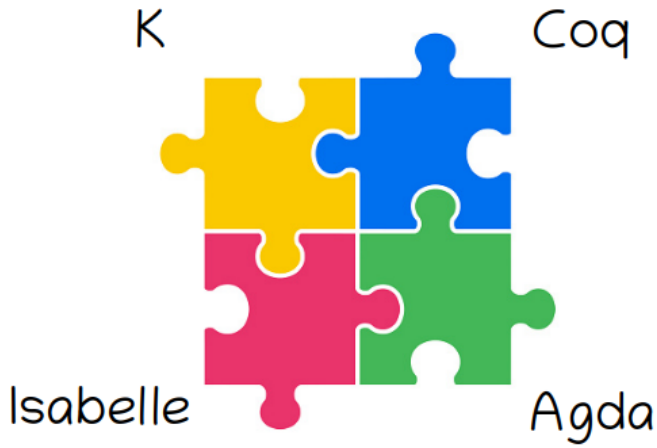
- **Objectives:**
  1. Translate a $\mathbb{K}$ semantics into Dedukti, to execute it.
  2. Translate a $\mathbb{K}$ semantics into Dedukti, to check proofs.
     → See my talk at Dedukti school!
  3. Export $\mathbb{K}$ semantics into Coq, Agda, etc. thanks to Dedukti.



- **Milestones:**
  1. Test the semantics.
  2. Recheck KProver proofs.
     → Work in progress:
     See my talk at WEPN!
  3. Prove properties about a language $\mathcal{L}$ described in $\mathbb{K}$.

Further work

- **Objectives:**
  1. Translate a $\mathbb{K}$ semantics into Dedukti, to execute it.
  2. Translate a $\mathbb{K}$ semantics into Dedukti, to check proofs.
     → See my talk at Dedukti school!
  3. Export $\mathbb{K}$ semantics into Coq, Agda, etc. thanks to Dedukti.



- **Milestones:**
  1. Test the semantics.
  2. Recheck KProver proofs.
     → Work in progress:
     See my talk at WEPN!
  3. Prove properties about a language $\mathcal{L}$ described in $\mathbb{K}$.
  4. Allow multi-formalism semantics.

Overview of Dedukti

- Symbols declaration:
  - Constant (`constant symbol`)
  - Defined (`symbol`)
- Typing thanks to $\lambda\Pi\equiv_{\mathcal{T}}$:
  - λ: Abstraction
  - Π: Dependent product
  - $\rightarrow$: Non-dependent product
  - `TYPE`
- Rewriting rule: `rule` *LHS* $\hookrightarrow$ *RHS*
- Example:

```
constant symbol Nat : TYPE;
constant symbol 0 : Nat;
constant symbol S : Nat → Nat;
symbol + : Nat → Nat → Nat;
rule $m + 0 ↪ $m;
rule $m + (S $n) ↪ S ($m + $n);
```

Overview of Matching Logic and Kore

- Matching Logic defines patterns φ.
  - $\varphi ::= x \mid X \mid \sigma \mid \varphi\ \varphi \mid \bot \mid \varphi \rightarrow \varphi \mid \exists x.\varphi \mid \mu X.\varphi$
  - A pattern is interpreted as the set of elements that it matches.

- Kore is a theory of Matching Logic
  - $=$ Theory of sorts
    $+$ Theory of rewriting
    $+$ Theory of equality

Extension of $\mathcal{L}_{MiniExp}$: $\mathcal{L}_{\mathrm{IMP}}$

`syntax` **AExp** ::= `Int` | `Id` | **AExp** "/" **AExp** > **AExp** "+" **AExp** | "(" **AExp** ")"	 [`left`, `strict`] [`left`, `strict`] [`bracket`]
`syntax` **BExp** ::= `Bool` | **AExp** "<" **AExp** | "not" **BExp** > **BExp** "and" **BExp** | "(" **BExp** ")"	 [`seqstrict`] [`strict`] [`left`, `strict(1)`] [`bracket`]

Extension of $\mathcal{L}_{MiniExp}$: $\mathcal{L}_{\text{IMP}}$

`syntax` **AExp** ::= `Int` | `Id` | **AExp** "/" **AExp** > **AExp** "+" **AExp** | "(" **AExp** ")"	 [`left`, `strict`] [`left`, `strict`] [`bracket`]
`syntax` **BExp** ::= `Bool` | **AExp** "<" **AExp** | "`not`" **BExp** > **BExp** "`and`" **BExp** | "(" **BExp** ")"	 [`seqstrict`] [`strict`] [`left`, `strict(1)`] [`bracket`]

- There are 2 ways to define an evaluation strategy:
  - Context
  - Attributes `strict` and `seqstrict`

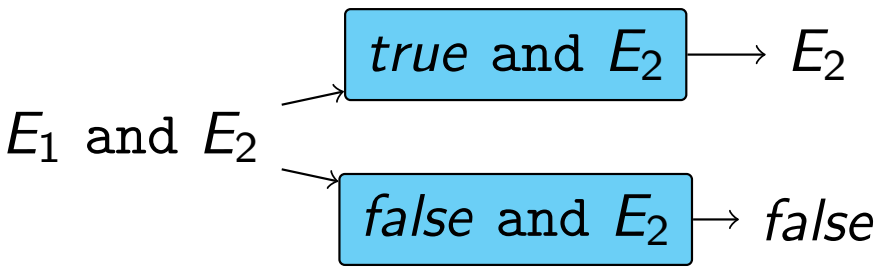
→ Everything is compiled in the same mechanism based on rewriting rules.

Evaluation strategy

- **Example**: **BExp** "`and`" **BExp** [`left`, `strict(1)`]

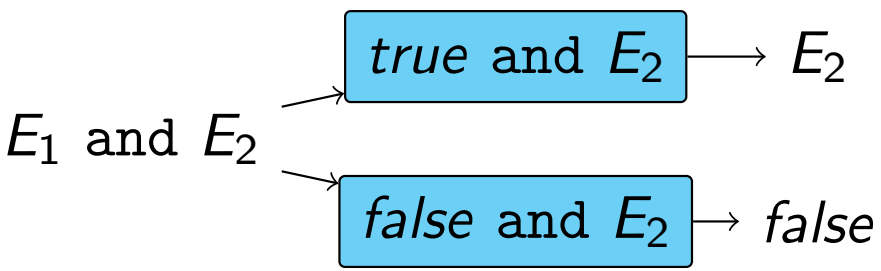
Evaluation strategy

- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]



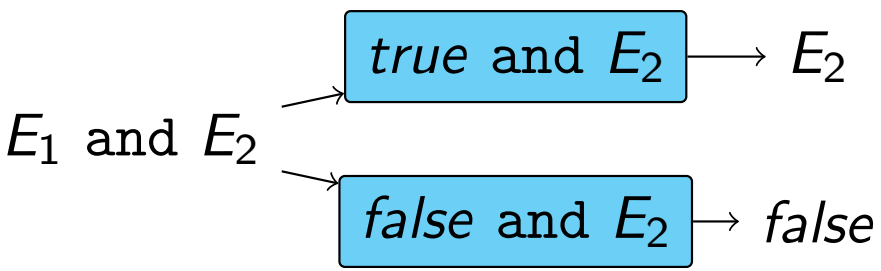
Evaluation strategy

- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]
  1. `rule` E_1 and E_2 `=>` $E_1 \curvearrowright (\clubsuit^1_{\text{and}}\ E_2)$ `requires` $E_1 \notin$ `KResult`
  2. `rule` $E_1 \curvearrowright (\clubsuit^1_{\text{and}}\ E_2)$ `=>` E_1 and E_2 `requires` $E_1 \in$ `KResult`



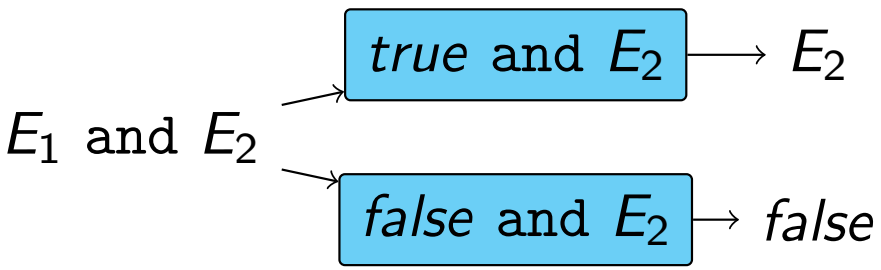
Evaluation strategy

- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]
  1. `rule` E_1 and E_2 `=>` $E_1 \curvearrowright (\clubsuit^1_{\text{and}}\ E_2)$ `requires` $E_1 \notin$ `KResult`
  2. `rule` $E_1 \curvearrowright (\clubsuit^1_{\text{and}}\ E_2)$ `=>` E_1 and E_2 `requires` $E_1 \in$ `KResult`
  3. `rule` *true* and b `=>` b
  4. `rule` *false* and _ `=>` *false*



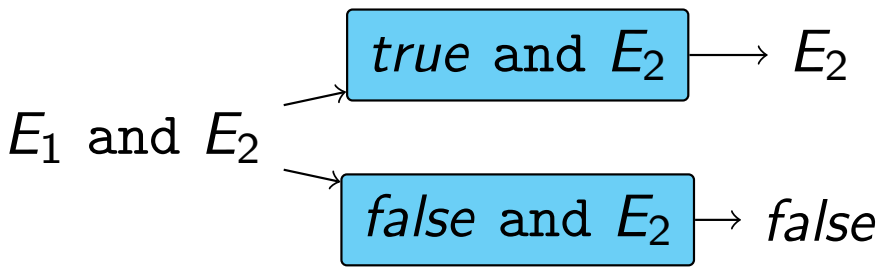
Evaluation strategy

- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]
  1. `rule` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$
     `=>` $\langle\ E_1 \curvearrowright (\boxplus^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$ `requires` $E_1 \notin$ `KResult`
  2. `rule` $E_1 \curvearrowright (\boxplus^1_{\text{and}}\ E_2)$ `=>` E_1 and E_2 `requires` $E_1 \in$ `KResult`
  3. `rule` *true* and b `=>` b
  4. `rule` *false* and _ `=>` *false*



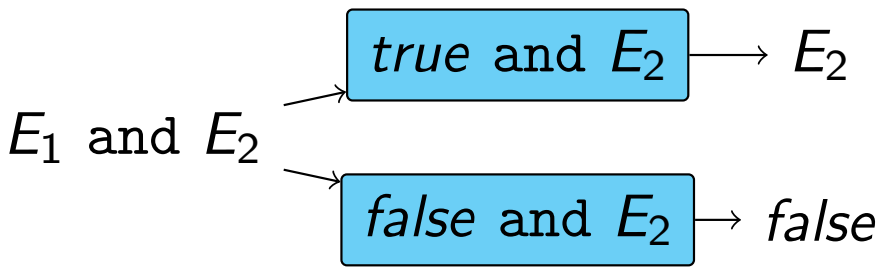
Evaluation strategy

- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]
  1. `rule` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$
     `=>` $\langle\ E_1 \curvearrowright (\clubsuit^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$ `requires` $E_1 \notin$ `KResult`
  2. `rule` $\langle\ E_1 \curvearrowright (\clubsuit^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$
     `=>` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$ `requires` $E_1 \in$ `KResult`
  3. `rule` *true* and b `=>` b
  4. `rule` *false* and _ `=>` *false*



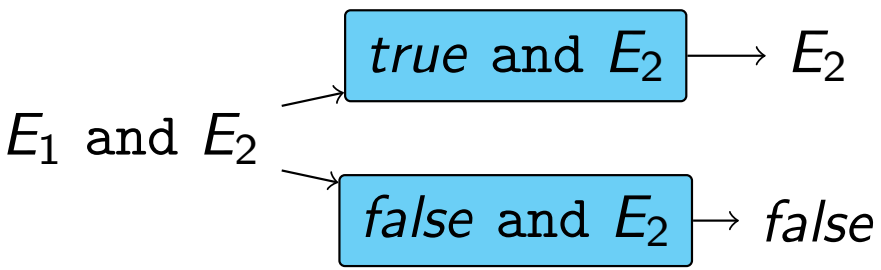
Evaluation strategy

- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]
  1. `rule` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$
     `=>` $\langle\ E_1 \curvearrowright (\freezeone{}{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$ `requires` $E_1 \notin$ `KResult`
  2. `rule` $\langle\ E_1 \curvearrowright (\freezeone{}{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$
     `=>` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$ `requires` $E_1 \in$ `KResult`
  3. `rule` $\langle\ \mathit{true}$ and $b \curvearrowright s\ \rangle_k m$ `=>` $\langle\ b \curvearrowright s\ \rangle_k m$
  4. `rule` *false* and _ `=>` *false*



Evaluation strategy

- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]
  1. `rule` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$
     `=>` $\langle\ E_1 \curvearrowright (\freezeone{}{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$ `requires` $E_1 \notin$ `KResult`
  2. `rule` $\langle\ E_1 \curvearrowright (\freezeone{}{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$
     `=>` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$ `requires` $E_1 \in$ `KResult`
  3. `rule` $\langle\ \mathit{true}$ and $b \curvearrowright s\ \rangle_k m$ `=>` $\langle\ b \curvearrowright s\ \rangle_k m$
  4. `rule` $\langle\ \mathit{false}$ and $\_ \curvearrowright s\ \rangle_k m$ `=>` $\langle\ \mathit{false} \curvearrowright s\ \rangle_k m$



Evaluation strategy

- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]
  1. `rule` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$
     `=>` $\langle\ E_1 \curvearrowright (\freezebox{1}{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$ `requires` $E_1 \notin$ `KResult`
  2. `rule` $\langle\ E_1 \curvearrowright (\freezebox{1}{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$
     `=>` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$ `requires` $E_1 \in$ `KResult`
  3. `rule` $\langle\ \mathit{true}$ and $b \curvearrowright s\ \rangle_k m$ `=>` $\langle\ b \curvearrowright s\ \rangle_k m$
  4. `rule` $\langle\ \mathit{false}$ and _ $\curvearrowright s\ \rangle_k m$ `=>` $\langle\ \mathit{false} \curvearrowright s\ \rangle_k m$

  $\langle\ (\mathit{true}$ `and` $\mathit{false})$ and $(\mathit{true}$ and $\mathit{true}) \curvearrowright .\ \rangle_k$`.Map`

Evaluation strategy

- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]

1. `rule` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$
   `=>` $\langle\ E_1 \curvearrowright (\freezeone{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$ `requires` $E_1 \notin$ `KResult`
2. `rule` $\langle\ E_1 \curvearrowright (\freezeone{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$
   `=>` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$ `requires` $E_1 \in$ `KResult`
3. `rule` $\langle\ \mathit{true}$ and $b \curvearrowright s\ \rangle_k m$ `=>` $\langle\ b \curvearrowright s\ \rangle_k m$
4. `rule` $\langle\ \mathit{false}$ and $\_ \curvearrowright s\ \rangle_k m$ `=>` $\langle\ \mathit{false} \curvearrowright s\ \rangle_k m$

$\langle\ (\mathit{true}$ and $\mathit{false})$ and $(\mathit{true}$ and $\mathit{true}) \curvearrowright .\ \rangle_k$ `.Map`
$\hookrightarrow_1 \langle\ (\mathit{true}$ and $\mathit{false}) \curvearrowright (\freezeone{\text{and}}\ (\mathit{true}$ and $\mathit{true})) \curvearrowright .\ \rangle_k$ `.Map`

Evaluation strategy

- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]

1. `rule` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$
   `=>` $\langle\ E_1 \curvearrowright (\freezeone{1}_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$ `requires` $E_1 \notin$ `KResult`
2. `rule` $\langle\ E_1 \curvearrowright (\freezeone{1}_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$
   `=>` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$ `requires` $E_1 \in$ `KResult`
3. `rule` $\langle\ \mathit{true}$ and $b \curvearrowright s\ \rangle_k m$ `=>` $\langle\ b \curvearrowright s\ \rangle_k m$
4. `rule` $\langle\ \mathit{false}$ and _ $\curvearrowright s\ \rangle_k m$ `=>` $\langle\ \mathit{false} \curvearrowright s\ \rangle_k m$

$\langle\ (\mathit{true}$ and $\mathit{false})$ and $(\mathit{true}$ and $\mathit{true}) \curvearrowright .\ \rangle_k$ `.Map`
$\hookrightarrow_1 \langle\ (\mathit{true}$ and $\mathit{false}) \curvearrowright (\freezeone{1}_{\text{and}}\ (\mathit{true}$ and $\mathit{true})) \curvearrowright .\ \rangle_k$ `.Map`
$\hookrightarrow_3 \langle\ \mathit{false} \curvearrowright (\freezeone{1}_{\text{and}}\ (\mathit{true}$ and $\mathit{true})) \curvearrowright .\ \rangle_k$ `.Map`

Evaluation strategy

- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]

1. `rule` $\langle\ E_1 \text{ and } E_2 \curvearrowright s\ \rangle_k m$
   `=>` $\langle\ E_1 \curvearrowright (\clubsuit^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$ `requires` $E_1 \notin$ `KResult`
2. `rule` $\langle\ E_1 \curvearrowright (\clubsuit^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$
   `=>` $\langle\ E_1 \text{ and } E_2 \curvearrowright s\ \rangle_k m$ `requires` $E_1 \in$ `KResult`
3. `rule` $\langle\ \mathit{true} \text{ and } b \curvearrowright s\ \rangle_k m$ `=>` $\langle\ b \curvearrowright s\ \rangle_k m$
4. `rule` $\langle\ \mathit{false} \text{ and } \_ \curvearrowright s\ \rangle_k m$ `=>` $\langle\ \mathit{false} \curvearrowright s\ \rangle_k m$

$\langle\ (\mathit{true} \text{ and } \mathit{false}) \text{ and } (\mathit{true} \text{ and } \mathit{true}) \curvearrowright .\ \rangle_k$ `.Map`
$\hookrightarrow_1 \langle\ (\mathit{true} \text{ and } \mathit{false}) \curvearrowright (\clubsuit^1_{\text{and}}\ (\mathit{true} \text{ and } \mathit{true})) \curvearrowright .\ \rangle_k$ `.Map`
$\hookrightarrow_3 \langle\ \mathit{false} \curvearrowright (\clubsuit^1_{\text{and}}\ (\mathit{true} \text{ and } \mathit{true})) \curvearrowright .\ \rangle_k$ `.Map`
$\hookrightarrow_2 \langle\ \mathit{false} \text{ and } (\mathit{true} \text{ and } \mathit{true}) \curvearrowright .\ \rangle_k$ `.Map`

Evaluation strategy

- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]

1. `rule` $\langle\ E_1 \text{ and } E_2 \curvearrowright s\ \rangle_k m$
   `=>` $\langle\ E_1 \curvearrowright (\boxplus^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$ `requires` $E_1 \notin$ `KResult`
2. `rule` $\langle\ E_1 \curvearrowright (\boxplus^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$
   `=>` $\langle\ E_1 \text{ and } E_2 \curvearrowright s\ \rangle_k m$ `requires` $E_1 \in$ `KResult`
3. `rule` $\langle\ \mathit{true} \text{ and } b \curvearrowright s\ \rangle_k m$ `=>` $\langle\ b \curvearrowright s\ \rangle_k m$
4. `rule` $\langle\ \mathit{false} \text{ and } \_ \curvearrowright s\ \rangle_k m$ `=>` $\langle\ \mathit{false} \curvearrowright s\ \rangle_k m$

$$\langle\ (\mathit{true} \text{ and } \mathit{false}) \text{ and } (\mathit{true} \text{ and } \mathit{true}) \curvearrowright .\ \rangle_k \texttt{.Map}$$
$$\hookrightarrow_1 \langle\ (\mathit{true} \text{ and } \mathit{false}) \curvearrowright (\boxplus^1_{\text{and}}\ (\mathit{true} \text{ and } \mathit{true})) \curvearrowright .\ \rangle_k \texttt{.Map}$$
$$\hookrightarrow_3 \langle\ \mathit{false} \curvearrowright (\boxplus^1_{\text{and}}\ (\mathit{true} \text{ and } \mathit{true})) \curvearrowright .\ \rangle_k \texttt{.Map}$$
$$\hookrightarrow_2 \langle\ \mathit{false} \text{ and } (\mathit{true} \text{ and } \mathit{true}) \curvearrowright .\ \rangle_k \texttt{.Map}$$
$$\hookrightarrow_4 \langle\ \mathit{false} \curvearrowright .\ \rangle_k \texttt{.Map}$$

Evaluation strategy

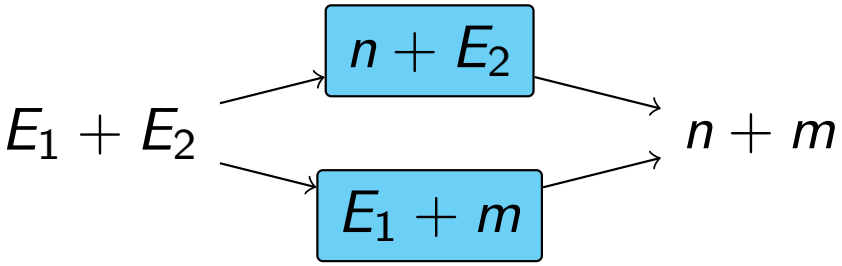
- **Example**: **BExp** "and" **BExp** [`left`, `strict(1)`]
  1. `rule` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$
     `=>` $\langle\ E_1 \curvearrowright (\overset{1}{\curlywedge}_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$ `requires` $E_1 \notin$ `KResult`
  2. `rule` $\langle\ E_1 \curvearrowright (\overset{1}{\curlywedge}_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k m$
     `=>` $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k m$ `requires` $E_1 \in$ `KResult`
  3. `rule` $\langle\ \mathit{true}$ and $b \curvearrowright s\ \rangle_k m$ `=>` $\langle\ b \curvearrowright s\ \rangle_k m$
  4. `rule` $\langle\ \mathit{false}$ and $\_ \curvearrowright s\ \rangle_k m$ `=>` $\langle\ \mathit{false} \curvearrowright s\ \rangle_k m$

  $\langle\ (\mathit{true}$ and $\mathit{false})$ and $(\mathit{true}$ and $\mathit{true}) \curvearrowright .\ \rangle_k$`.Map`
  $\hookrightarrow_1 \langle\ (\mathit{true}$ and $\mathit{false}) \curvearrowright (\overset{1}{\curlywedge}_{\text{and}}\ (\mathit{true}$ and $\mathit{true})) \curvearrowright .\ \rangle_k$`.Map`
  $\hookrightarrow_3 \langle\ \mathit{false} \curvearrowright (\overset{1}{\curlywedge}_{\text{and}}\ (\mathit{true}$ and $\mathit{true})) \curvearrowright .\ \rangle_k$`.Map`
  $\hookrightarrow_2 \langle\ \mathit{false}$ and $(\mathit{true}$ and $\mathit{true}) \curvearrowright .\ \rangle_k$`.Map`
  $\hookrightarrow_4 \langle\ \mathit{false} \curvearrowright .\ \rangle_k$`.Map`

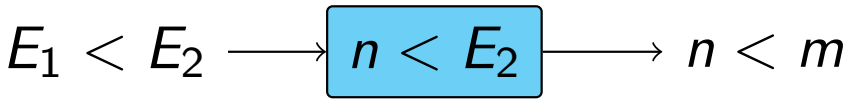
- **K computation**: a list (`List{`K`,` $\curvearrowright$`}`), potentially nested, of computations to be performed sequentially.
- **The sort** `KResult`, to distinguish the (final) values of expressions (Here, `syntax KResult ::= Int | Bool`).

Evaluation strategy

Attribute `strict`

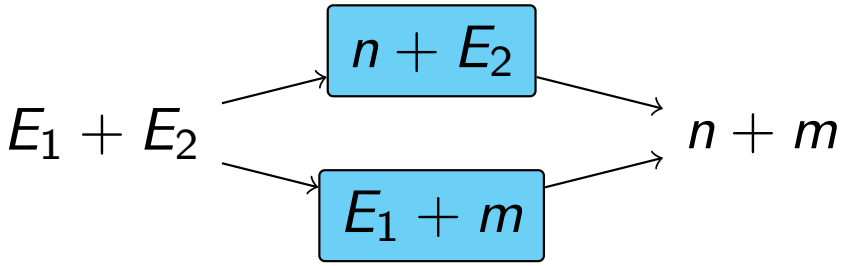


Attribute `seqstrict`



Evaluation strategy

Attribute `strict`

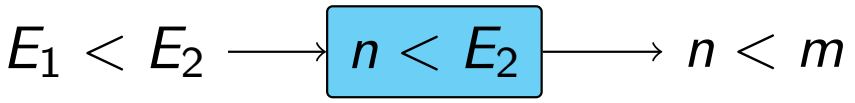


$$\texttt{rule}\ E_1 + E_2 \Rightarrow E_1 \curvearrowright (\circledast^1_+\ E_2)$$
$$\texttt{requires}\ E_1 \notin \texttt{KResult}$$
$$\texttt{rule}\ E_1 \curvearrowright (\circledast^1_+\ E_2) \Rightarrow E_1 + E_2$$
$$\texttt{requires}\ E_1 \in \texttt{KResult}$$

$$\texttt{rule}\ E_1 + E_2 \Rightarrow E_2 \curvearrowright (\circledast^2_+\ E_1)$$
$$\texttt{requires}\ E_2 \notin \texttt{KResult}$$

$$\texttt{rule}\ E_2 \curvearrowright (\circledast^2_+\ E_1) \Rightarrow E_1 + E_2$$
$$\texttt{requires}\ E_2 \in \texttt{KResult}$$

Attribute `seqstrict`



$$\texttt{rule}\ E_1 < E_2 \Rightarrow E_1 \curvearrowright (\circledast^1_<\ E_2)$$
$$\texttt{requires}\ E_1 \notin \texttt{KResult}$$
$$\texttt{rule}\ E_1 \curvearrowright (\circledast^1_<\ E_2) \Rightarrow E_1 < E_2$$
$$\texttt{requires}\ E_1 \in \texttt{KResult}$$

$$\texttt{rule}\ E_1 < E_2 \Rightarrow E_2 \curvearrowright (\circledast^2_<\ E_1)$$
$$\texttt{requires}\ E_2 \notin \texttt{KResult} \wedge E_1 \in \texttt{KResult}$$

$$\texttt{rule}\ E_2 \curvearrowright (\circledast^2_<\ E_1) \Rightarrow E_1 < E_2$$
$$\texttt{requires}\ E_2 \in \texttt{KResult}$$