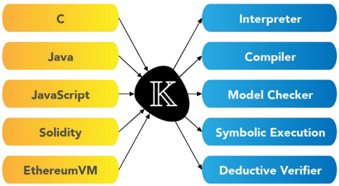
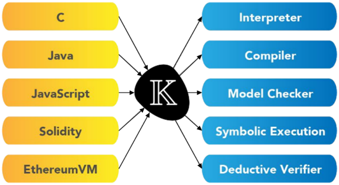
$\mathbb{K}$ framework in a nutshell

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



𝕂 framework in a nutshell

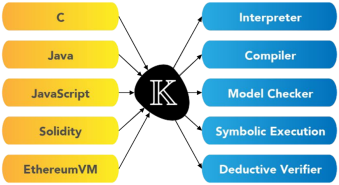
- Semantical framework
  - to define formal semantics of programming languages
  - to automatically generate tools from these semantics



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

$\mathbb{K}$ framework in a nutshell

- 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

- 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	✓	✗

Define a semantics with $\mathbb{K}$

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

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

Define a semantics with $\mathbb{K}$

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

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

Define a semantics with $\mathbb{K}$

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)

Define a semantics with $\mathbb{K}$

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

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

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

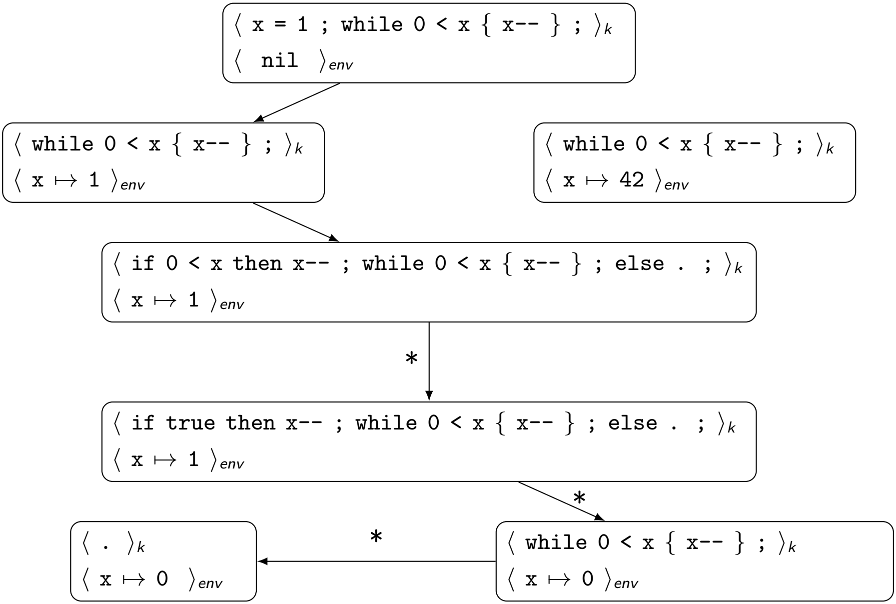
$$\langle\ \texttt{if 0 < x then x-- ; while 0 < x \{ x-- \} ; else . ;}\ \rangle_k$$
$$\langle\ \texttt{x} \mapsto 1\ \rangle_{env}$$

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

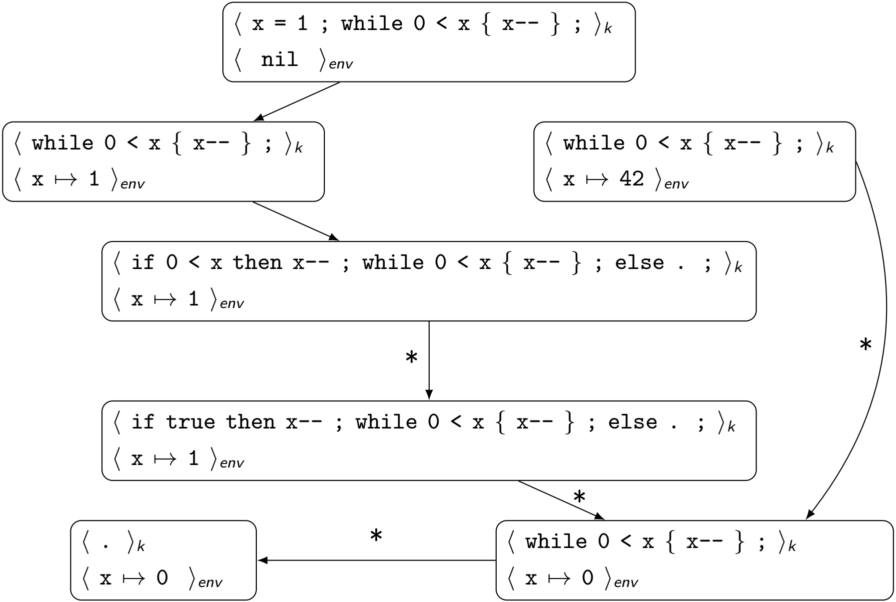
$$\langle\ \texttt{.}\ \rangle_k$$
$$\langle\ \texttt{x} \mapsto 0\ \rangle_{env}$$

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

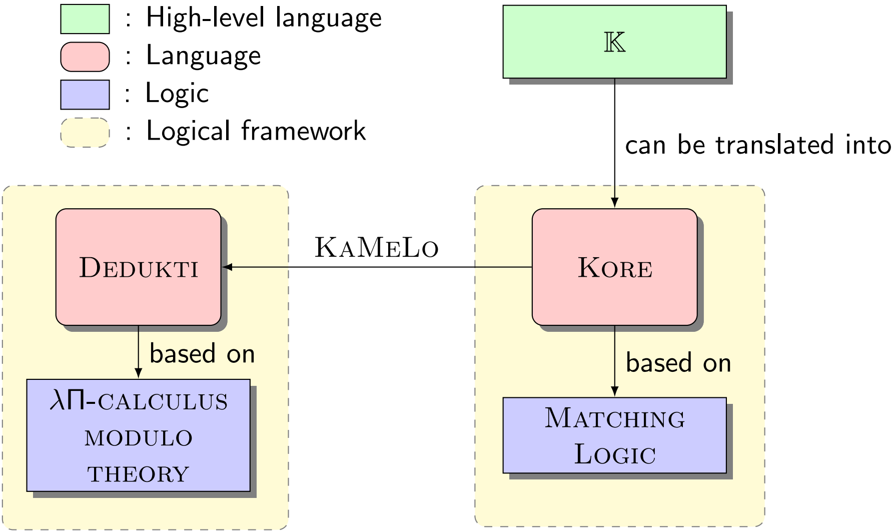
Define a semantics with 𝕂



Define a semantics with $\mathbb{K}$



Pipeline of the translation



Translate $\mathbb{K}$ into DEDUKTI

How to do it?

~~**How to do it?**~~

- **What is the purpose of the translation?**

Translate $\mathbb{K}$ into DEDUKTI

~~How to do it?~~

- **What is the purpose of the translation?**
- **What do we want to do with the result of the translation?**
  - Execute a program? $\rightsquigarrow$ shallow encoding
  - Check a proof? $\rightsquigarrow$ deep encoding

1. A shallow encoding to execute a program in Dedukti

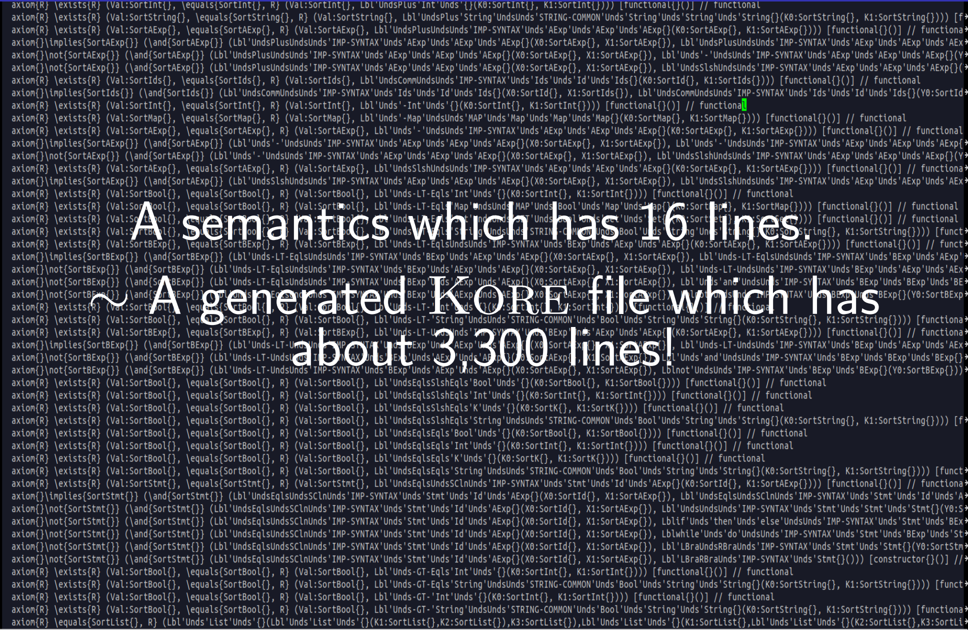
2. A deep encoding to check proofs in Dedukti
   - Translate Matching Logic constructors, notations and symbols
   - Translate Matching Logic proof system

3. Conclusion

KORE: A MATCHING LOGIC theory

```
axiom{R} \exists{R} (Val:SortInt{}, \equals{SortInt{}, R} (Val:SortInt{}, Lbl'UndsPlus'Int'Unds'{}(K0:SortInt{}, K1:SortInt{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortString{}, \equals{SortString{}, R} (Val:SortString{}, Lbl'UndsPlus'String'UndsUnds'STRING-COMMON'Unds'String'Unds'String'Unds'String{}(K0:SortString{}, K1:SortString{}))) [f
axiom{R} \exists{R} (Val:SortAExp{}, \equals{SortAExp{}, R} (Val:SortAExp{}, Lbl'UndsPlusUndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{}(K0:SortAExp{}, K1:SortAExp{}))) [functional{}()] // functiona
axiom{}\implies{SortAExp{}} (\and{SortAExp{}} (Lbl'UndsPlusUndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lbl'UndsPlusUndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AEx
axiom{}\not{SortAExp{}} (\and{SortAExp{}} (Lbl'UndsPlusUndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lbl'Unds'-'UndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{}(Y
axiom{}\not{SortAExp{}} (\and{SortAExp{}} (Lbl'UndsPlusUndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lbl'UndsSlshUndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{}(
axiom{R} \exists{R} (Val:SortIds{}, \equals{SortIds{}, R} (Val:SortIds{}, Lbl'UndsCommUndsUnds'IMP-SYNTAX'Unds'Ids'Unds'Id'Unds'Ids{}(K0:SortId{}, K1:SortIds{}))) [functional{}()] // functional
axiom{}\implies{SortIds{}} (\and{SortIds{}} (Lbl'UndsCommUndsUnds'IMP-SYNTAX'Unds'Ids'Unds'Id'Unds'Ids{}(X0:SortId{}, X1:SortIds{}), Lbl'UndsCommUndsUnds'IMP-SYNTAX'Unds'Ids'Unds'Id'Unds'Ids{}(Y0:SortId
axiom{R} \exists{R} (Val:SortInt{}, \equals{SortInt{}, R} (Val:SortInt{}, Lbl'Unds'-Int'Unds'{}(K0:SortInt{}, K1:SortInt{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortMap{}, \equals{SortMap{}, R} (Val:SortMap{}, Lbl'Unds'-Map'UndsUnds'MAP'Unds'Map'Unds'Map'Unds'Map{}(K0:SortMap{}, K1:SortMap{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortAExp{}, \equals{SortAExp{}, R} (Val:SortAExp{}, Lbl'Unds'-'UndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{}(K0:SortAExp{}, K1:SortAExp{}))) [functional{}()] // functional
axiom{}\implies{SortAExp{}} (\and{SortAExp{}} (Lbl'Unds'-'UndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lbl'Unds'-'UndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{
axiom{}\not{SortAExp{}} (\and{SortAExp{}} (Lbl'Unds'-'UndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lbl'UndsSlshUndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{}(Y
axiom{R} \exists{R} (Val:SortAExp{}, \equals{SortAExp{}, R} (Val:SortAExp{}, Lbl'UndsSlshUndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{}(K0:SortAExp{}, K1:SortAExp{}))) [functional{}()] // functiona
axiom{}\implies{SortAExp{}} (\and{SortAExp{}} (Lbl'UndsSlshUndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lbl'UndsSlshUndsUnds'IMP-SYNTAX'Unds'AExp'Unds'AExp'Unds'AEx
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'Unds-LT-Eqls'Int'Unds'{}(K0:SortInt{}, K1:SortInt{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'Unds-LT-Eqls'Map'UndsUnds'MAP'Unds'Bool'Unds'Map'Unds'Map{}(K0:SortMap{}, K1:SortMap{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'Unds-LT-Eqls'Set'UndsUnds'SET'Unds'Bool'Unds'Set'Unds'Set{}(K0:SortSet{}, K1:SortSet{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'Unds-LT-Eqls'String'UndsUnds'STRING-COMMON'Unds'Bool'Unds'String'Unds'String{}(K0:SortString{}, K1:SortString{}))) [funct
axiom{R} \exists{R} (Val:SortBExp{}, \equals{SortBExp{}, R} (Val:SortBExp{}, Lbl'Unds-LT-EqlsUndsUnds'IMP-SYNTAX'Unds'BExp'Unds'AExp'Unds'AExp{}(K0:SortAExp{}, K1:SortAExp{}))) [functional{}()] // funct
axiom{}\implies{SortBExp{}} (\and{SortBExp{}} (Lbl'Unds-LT-EqlsUndsUnds'IMP-SYNTAX'Unds'BExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lbl'Unds-LT-EqlsUndsUnds'IMP-SYNTAX'Unds'BExp'Unds'AExp'
axiom{}\not{SortBExp{}} (\and{SortBExp{}} (Lbl'Unds-LT-EqlsUndsUnds'IMP-SYNTAX'Unds'BExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lbl'Unds-LT-UndsUnds'IMP-SYNTAX'Unds'BExp'Unds'AExp'Unds'AEx
axiom{}\not{SortBExp{}} (\and{SortBExp{}} (Lbl'Unds-LT-EqlsUndsUnds'IMP-SYNTAX'Unds'BExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lbl'Unds'and'UndsUnds'IMP-SYNTAX'Unds'BExp'Unds'BExp'Unds'BE
axiom{}\not{SortBExp{}} (\and{SortBExp{}} (Lbl'Unds-LT-EqlsUndsUnds'IMP-SYNTAX'Unds'BExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lblnot'UndsUnds'IMP-SYNTAX'Unds'BExp'Unds'BExp{}(Y0:SortBExp
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'Unds-LT-'Int'Unds'{}(K0:SortInt{}, K1:SortInt{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'Unds-LT-'String'UndsUnds'STRING-COMMON'Unds'Bool'Unds'String'Unds'String{}(K0:SortString{}, K1:SortString{}))) [functiona
axiom{R} \exists{R} (Val:SortBExp{}, \equals{SortBExp{}, R} (Val:SortBExp{}, Lbl'Unds-LT-UndsUnds'IMP-SYNTAX'Unds'BExp'Unds'AExp'Unds'AExp{}(K0:SortAExp{}, K1:SortAExp{}))) [functional{}()] // functiona
axiom{}\implies{SortBExp{}} (\and{SortBExp{}} (Lbl'Unds-LT-UndsUnds'IMP-SYNTAX'Unds'BExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lbl'Unds-LT-UndsUnds'IMP-SYNTAX'Unds'BExp'Unds'AExp'Unds'AEx
axiom{}\not{SortBExp{}} (\and{SortBExp{}} (Lbl'Unds-LT-UndsUnds'IMP-SYNTAX'Unds'BExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lbl'Unds'and'UndsUnds'IMP-SYNTAX'Unds'BExp'Unds'BExp'Unds'BExp{}
axiom{}\not{SortBExp{}} (\and{SortBExp{}} (Lbl'Unds-LT-UndsUnds'IMP-SYNTAX'Unds'BExp'Unds'AExp'Unds'AExp{}(X0:SortAExp{}, X1:SortAExp{}), Lblnot'UndsUnds'IMP-SYNTAX'Unds'BExp'Unds'BExp{}(Y0:SortBExp{}))
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'UndsEqlsSlshEqls'Bool'Unds'{}(K0:SortBool{}, K1:SortBool{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'UndsEqlsSlshEqls'Int'Unds'{}(K0:SortInt{}, K1:SortInt{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'UndsEqlsSlshEqls'K'Unds'{}(K0:SortK{}, K1:SortK{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'UndsEqlsSlshEqls'String'UndsUnds'STRING-COMMON'Unds'Bool'Unds'String'Unds'String{}(K0:SortString{}, K1:SortString{}))) [f
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'UndsEqlsEqls'Bool'Unds'{}(K0:SortBool{}, K1:SortBool{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'UndsEqlsEqls'Int'Unds'{}(K0:SortInt{}, K1:SortInt{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'UndsEqlsEqls'K'Unds'{}(K0:SortK{}, K1:SortK{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'UndsEqlsEqls'String'UndsUnds'STRING-COMMON'Unds'Bool'Unds'String'Unds'String{}(K0:SortString{}, K1:SortString{}))) [funct
axiom{R} \exists{R} (Val:SortStmt{}, \equals{SortStmt{}, R} (Val:SortStmt{}, Lbl'UndsEqlsUndsSClnUnds'IMP-SYNTAX'Unds'Stmt'Unds'Id'Unds'AExp{}(K0:SortId{}, K1:SortAExp{}))) [functional{}()] // functiona
axiom{}\implies{SortStmt{}} (\and{SortStmt{}} (Lbl'UndsEqlsUndsSClnUnds'IMP-SYNTAX'Unds'Stmt'Unds'Id'Unds'AExp{}(X0:SortId{}, X1:SortAExp{}), Lbl'UndsEqlsUndsSClnUnds'IMP-SYNTAX'Unds'Stmt'Unds'Id'Unds'A
axiom{}\not{SortStmt{}} (\and{SortStmt{}} (Lbl'UndsEqlsUndsSClnUnds'IMP-SYNTAX'Unds'Stmt'Unds'Id'Unds'AExp{}(X0:SortId{}, X1:SortAExp{}), Lbl'UndsUndsUnds'IMP-SYNTAX'Unds'Stmt'Unds'Stmt'Unds'Stmt{}(Y0:S
axiom{}\not{SortStmt{}} (\and{SortStmt{}} (Lbl'UndsEqlsUndsSClnUnds'IMP-SYNTAX'Unds'Stmt'Unds'Id'Unds'AExp{}(X0:SortId{}, X1:SortAExp{}), Lblif'Unds'then'Unds'else'UndsUnds'IMP-SYNTAX'Unds'Stmt'Unds'BEx
axiom{}\not{SortStmt{}} (\and{SortStmt{}} (Lbl'UndsEqlsUndsSClnUnds'IMP-SYNTAX'Unds'Stmt'Unds'Id'Unds'AExp{}(X0:SortId{}, X1:SortAExp{}), Lblwhile'Unds'do'UndsUnds'IMP-SYNTAX'Unds'Stmt'Unds'BExp'Unds'St
axiom{}\not{SortStmt{}} (\and{SortStmt{}} (Lbl'UndsEqlsUndsSClnUnds'IMP-SYNTAX'Unds'Stmt'Unds'Id'Unds'AExp{}(X0:SortId{}, X1:SortAExp{}), Lbl'LBraUndsRBraUnds'IMP-SYNTAX'Unds'Stmt'Unds'Stmt{}(Y0:SortStm
axiom{}\not{SortStmt{}} (\and{SortStmt{}} (Lbl'UndsEqlsUndsSClnUnds'IMP-SYNTAX'Unds'Stmt'Unds'Id'Unds'AExp{}(X0:SortId{}, X1:SortAExp{}), Lbl'LBraRBraUnds'IMP-SYNTAX'Unds'Stmt{}())) [constructor{}()] //
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'Unds-GT-Eqls'Int'Unds'{}(K0:SortInt{}, K1:SortInt{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'Unds-GT-Eqls'String'UndsUnds'STRING-COMMON'Unds'Bool'Unds'String'Unds'String{}(K0:SortString{}, K1:SortString{}))) [funct
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'Unds-GT-'Int'Unds'{}(K0:SortInt{}, K1:SortInt{}))) [functional{}()] // functional
axiom{R} \exists{R} (Val:SortBool{}, \equals{SortBool{}, R} (Val:SortBool{}, Lbl'Unds-GT-'String'UndsUnds'STRING-COMMON'Unds'Bool'Unds'String'Unds'String{}(K0:SortString{}, K1:SortString{}))) [functiona
axiom{R} \equals{SortList{}, R} (Lbl'Unds'List'Unds'{}(Lbl'Unds'List'Unds'{}(K1:SortList{},K2:SortList{}),K3:SortList{}),Lbl'Unds'List'Unds'{}(K1:SortList{},Lbl'Unds'List'Unds'{}(K2:SortList{},K3:SortLi
```

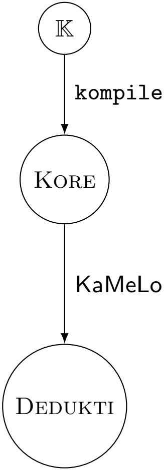
Kore: A Matching Logic theory



A semantics which has 16 lines.

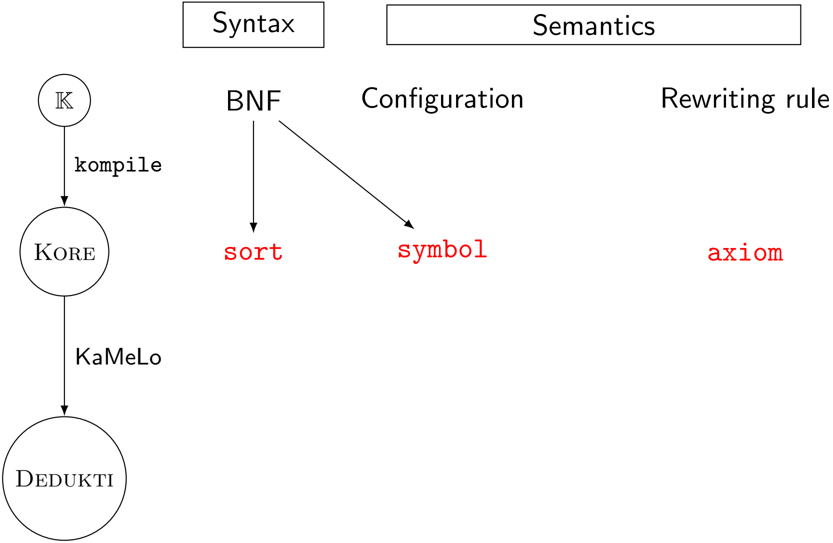
$\sim$ A generated Kore file which has about 3,300 lines!

Translation from $\mathbb{K}$ to DEDUKTI

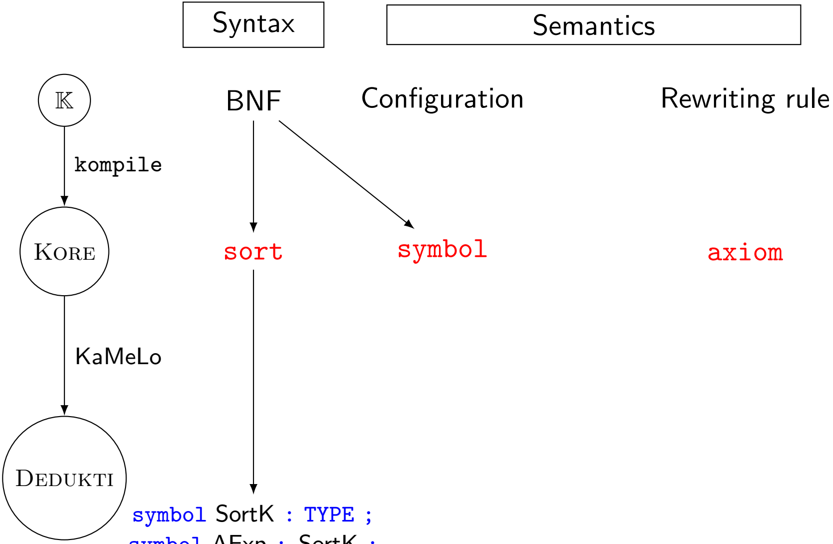


	Syntax	Semantics	
	BNF	Configuration	Rewriting rule
	`sort`	`symbol`	`axiom`

Translation from 𝕂 to DEDUKTI

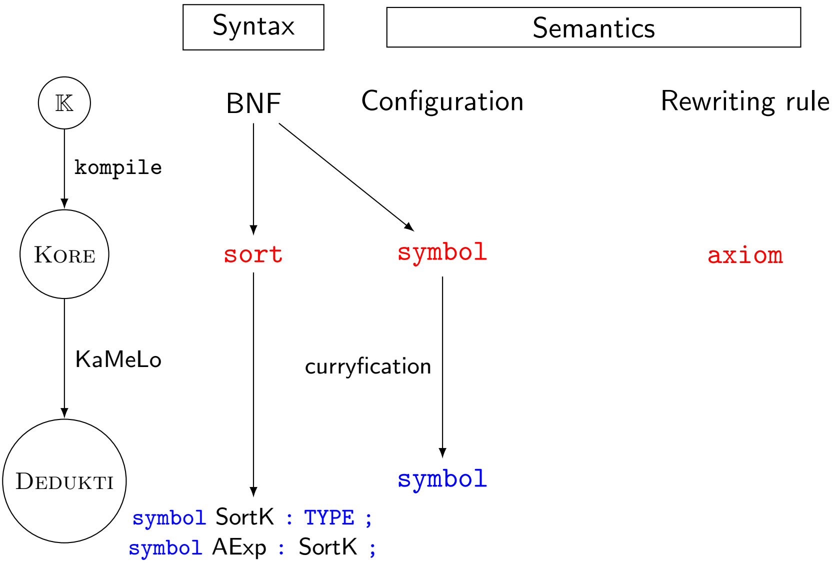


Translation from $\mathbb{K}$ to Dedukti

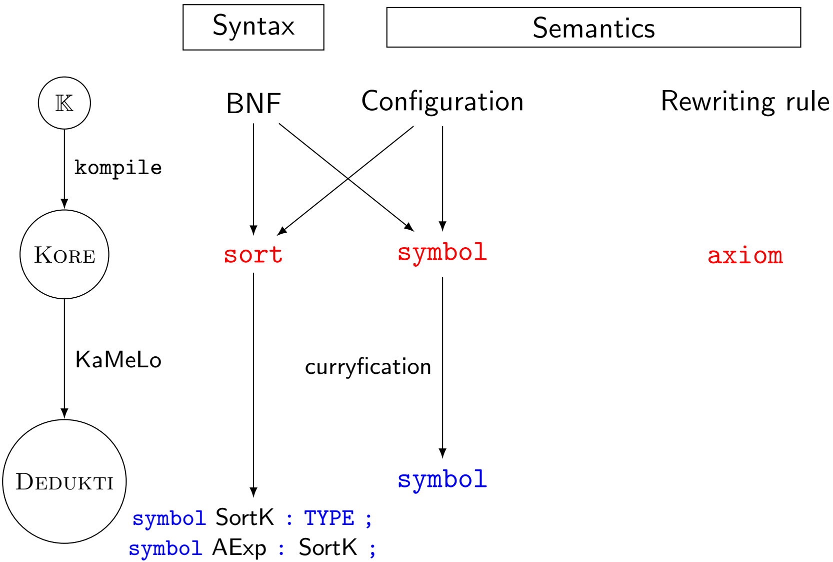



```
symbol SortK : TYPE ;
symbol AExp : SortK ;
```

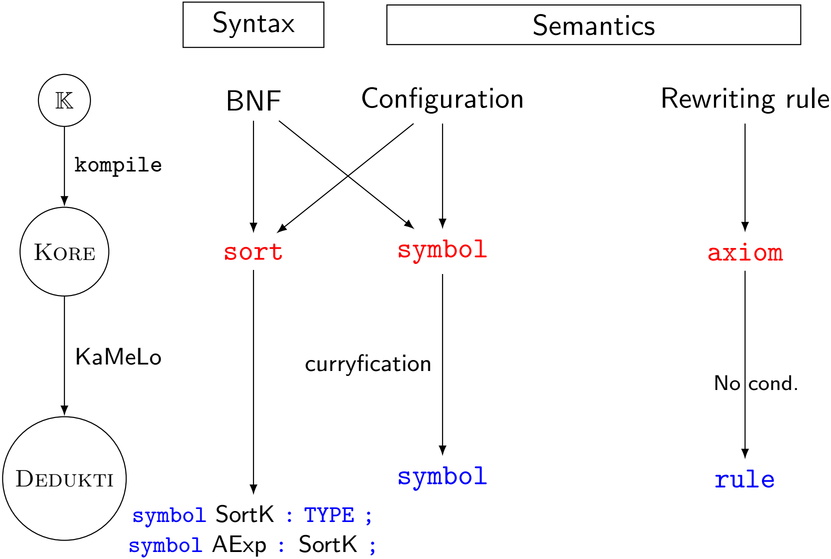
Translation from $\mathbb{K}$ to Dedukti



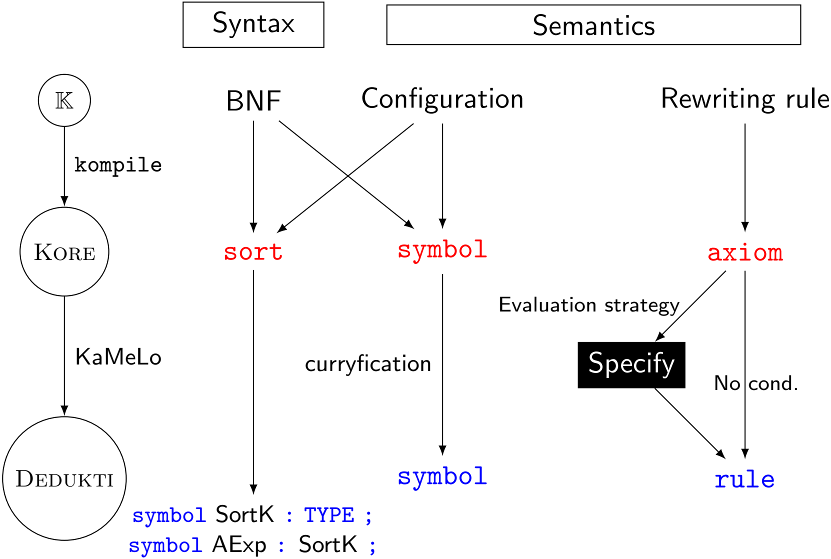
Translation from 𝕂 to DEDUKTI



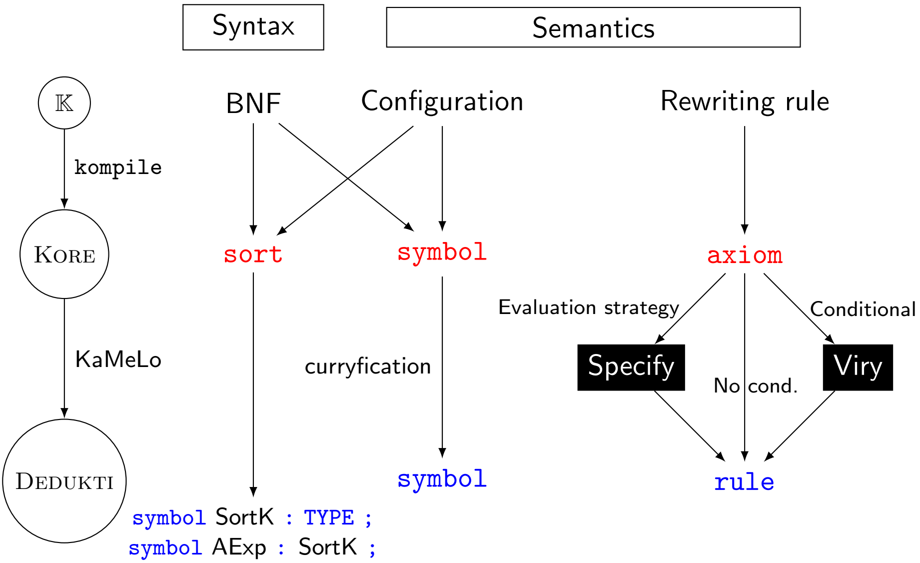
Translation from 𝕂 to Dedukti



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.**
- **Evaluated expressions have a specific type.**

Translate evaluation strategies

Generated rules to define evaluation strategies:

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

Translation into Dedukti:

1. Instantiation of E_1:

   a. rule $\langle\ (\mathit{not}\ \$X1)\ \mathit{and}\ \$E2 \curvearrowright \$s\ \rangle_k$
   $\hookrightarrow\ \langle\ (\mathit{not}\ \$X1) \curvearrowright (\circledast^1_{\text{and}}\ \$E2) \curvearrowright \$s\ \rangle_k$

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

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

The grammar of **BExp**:

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

Translate a CTRS to a TRS[1]

[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

[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] 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**:

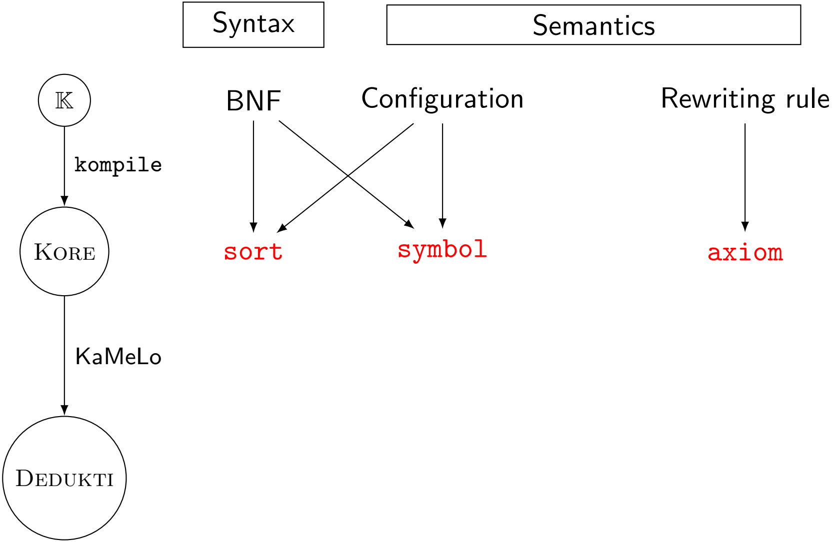
  (A) rule $max\ X\ Y$ => Y requires X <Int Y
  (B) rule $max\ X\ Y$ => X [owise]

  translated into

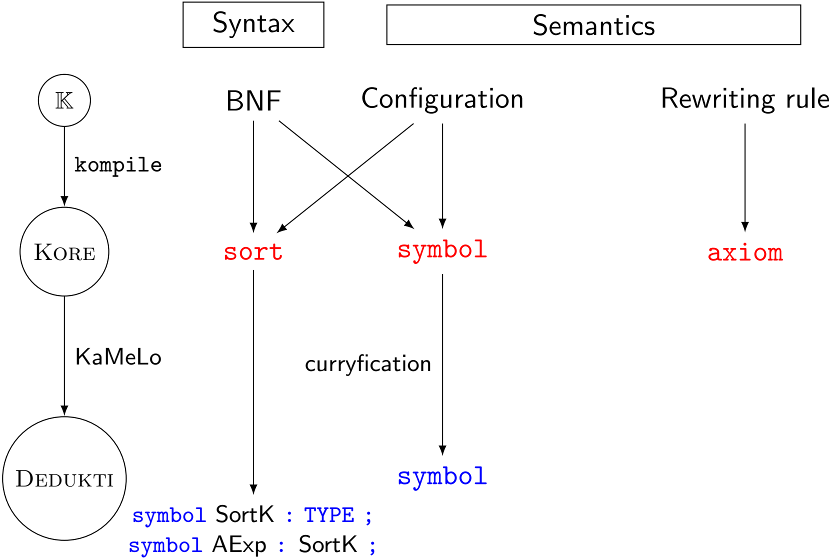
  (ℵ) rule max $x $y $\hookrightarrow$ $\flat max$ $x $y ($x $<$ $y)
  (A') rule $\flat max$ $x $y $true$ $\hookrightarrow$ $y
  (B') rule $\flat max$ $x $y $false$ $\hookrightarrow$ $x

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

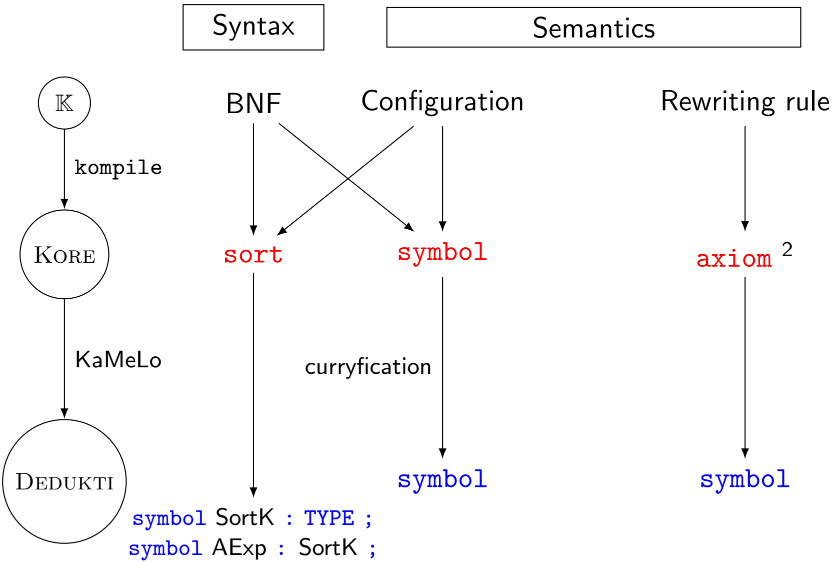
Translation from 𝕂 to Dedukti



Translation from 𝕂 to DEDUKTI



Translation from $\mathbb{K}$ to DEDUKTI



[2] MATCHING LOGIC pattern

1. A shallow encoding to execute a program in Dedukti

2. A deep encoding to check proofs in Dedukti
   - Translate Matching Logic constructors, notations and symbols
   - Translate Matching Logic proof system

3. Conclusion

Matching Logic constructors

Matching Logic defines patterns

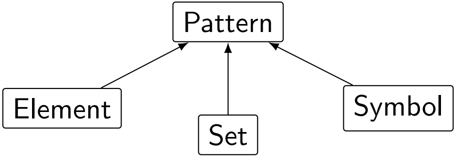
$\varphi ::= x \mid X \mid \sigma \mid \varphi \mathbin{@} \varphi \mid \bot \mid \varphi \rightarrow \varphi \mid \exists x.\varphi \mid \mu X.\varphi$

Matching Logic constructors

Matching Logic defines patterns

$\varphi ::= x \mid X \mid \sigma \mid \varphi \; @ \; \varphi \mid \bot \mid \varphi \rightarrow \varphi \mid \exists x.\varphi \mid \mu X.\varphi$

```
symbol #Pattern   : TYPE;
symbol #Element   : TYPE;
symbol #Set       : TYPE;
symbol #Symbol    : TYPE;
```



The next symbol:

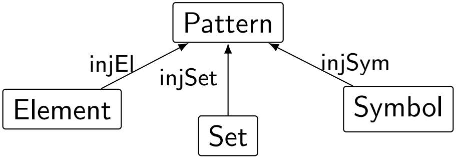
```
symbol • : #Symbol; // Symbol
```

Matching Logic constructors

Matching Logic defines patterns

$\varphi ::= x \mid X \mid \sigma \mid \varphi \,@\, \varphi \mid \bot \mid \varphi \rightarrow \varphi \mid \exists x.\varphi \mid \mu X.\varphi$

```
symbol #Pattern   : TYPE;
symbol #Element   : TYPE;
symbol #Set       : TYPE;
symbol #Symbol    : TYPE;
```



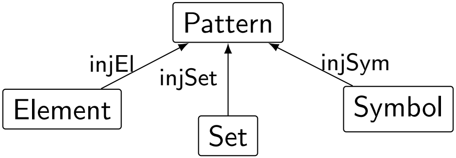

```
symbol injEl  : #Element → #Pattern;
symbol injSet : #Set → #Pattern;
symbol injSym : #Symbol → #Pattern;
```

Matching Logic constructors

Matching Logic defines patterns

$\varphi ::= x \mid X \mid \sigma \mid \varphi \mathbin{@} \varphi \mid \bot \mid \varphi \to \varphi \mid \exists x.\varphi \mid \mu X.\varphi$

```
symbol #Pattern  : TYPE;
symbol #Element  : TYPE;
symbol #Set      : TYPE;
symbol #Symbol   : TYPE;
```




```
symbol injEl  : #Element → #Pattern;
symbol injSet : #Set → #Pattern;
symbol injSym : #Symbol → #Pattern;
```

```
symbol @_ML  : #Pattern → #Pattern → #Pattern;
symbol ⊥_ML  : #Pattern;
symbol ⇒_ML  : #Pattern → #Pattern → #Pattern;
symbol ∃_ML  : (#Element → #Pattern) → #Pattern;
symbol μ_ML  : (#Set → #Pattern) → #Pattern;
```

Notations vs Symbols

- Notations are syntactic sugar:

```
symbol ¬_ML : #Pattern → #Pattern;
rule ¬_ML $φ ↪ $φ ⇒_ML ⊥_ML;

symbol ∨_ML : #Pattern → #Pattern → #Pattern;
rule $φ0 ∨_ML $φ1 ↪ (¬_ML $φ0) ⇒_ML $φ1;
```

- Symbols are patterns:

```
symbol • : #Symbol; // Symbol
symbol ⇝ : #Pattern → #Pattern → #Pattern; // Notation
rule $φ1 ⇝ $φ2 ↪ $φ1 ⇒_ML ((injSym •) @_ML $φ2);
```

FOL Reasoning

$$\frac{}{\varphi \to (\psi \to \varphi)}\ \text{(Prop 1)}$$

$$\frac{}{(\varphi \to (\psi \to \theta)) \to ((\varphi \to \psi) \to (\varphi \to \theta))}\ \text{(Prop 2)}$$

$$\frac{}{((\varphi \to \bot) \to \bot) \to \varphi}\ \text{(Prop 3)}$$

$$\frac{\varphi_1 \quad \varphi_1 \to \varphi_2}{\varphi_2}\ \text{(Modus Ponens)}$$

$$\frac{}{\varphi[y/x] \to \exists x.\varphi}\ \text{(}\exists\text{-Quantifier)}$$

$$\frac{\varphi_1 \to \varphi_2 \quad (when\ x \notin FV(\varphi_2))}{(\exists x.\varphi_1) \to \varphi_2}\ \text{(}\exists\text{-Generalization)}$$

Technical rules

$$\frac{}{\exists x.x}\ \text{(Existence)}$$

$$\frac{}{\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])}\ \text{(Singleton)}$$

Frame Reasoning

$$\frac{}{C[\bot] \to \bot}\ (Propagation_\bot)$$

$$\frac{}{C[\varphi_1 \vee \varphi_2] \to C[\varphi_1] \vee C[\varphi_2]}\ (Propagation_\vee)$$

$$\frac{(when\ x \notin FV(C))}{C[\exists x.\varphi] \to \exists x.C[\varphi]}\ (Propagation_\exists)$$

$$\frac{\varphi_1 \to \varphi_2}{C[\varphi_1] \to C[\varphi_2]}\ \text{(Framing)}$$

Fixpoint Reasoning

$$\frac{\varphi}{\varphi[\psi/X]}\ \text{(Set Variable Substitution)}$$

$$\frac{}{\varphi[(\mu X.\varphi/X)] \to \mu X.\varphi}\ \text{(PreFixpoint)}$$

$$\frac{\varphi[\psi/X] \to \psi}{\mu X.\varphi \to \psi}\ \text{(Knaster-Tarski)}$$

Propositional fragment

$$\frac{}{\varphi \rightarrow (\psi \rightarrow \varphi)} \text{(Prop 1)}$$

$$\frac{}{(\varphi \rightarrow (\psi \rightarrow \theta)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \theta))} \text{(Prop 2)}$$

$$\frac{}{((\varphi \rightarrow \bot) \rightarrow \bot) \rightarrow \varphi} \text{(Prop 3)}$$

$$\frac{\varphi_1 \quad \varphi_1 \rightarrow \varphi_2}{\varphi_2} \text{(Modus Ponens)}$$

$\rightarrow$ No difficulty!

As usual:

```
injective symbol Prf : #Pattern → TYPE;
```

Propositional fragment

- $$\frac{}{\varphi \ \rightarrow \ (\psi \ \rightarrow \ \varphi)} \text{ (Prop 1)}$$

```
symbol prop-1 : Π (φ ψ : #Pattern),
  Prf (φ ⇒_ML (ψ ⇒_ML φ));
```

Propositional fragment

- $$\frac{}{\varphi \rightarrow (\psi \rightarrow \varphi)} \text{(Prop 1)}$$

```
symbol prop-1 : Π (φ ψ : #Pattern),
  Prf (φ ⇒_ML (ψ ⇒_ML φ));
```

- $$\frac{}{(\varphi \rightarrow (\psi \rightarrow \theta)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \theta))} \text{(Prop 2)}$$

  $\rightarrow$ **To be done as an exercise!**

- $$\frac{}{((\varphi \rightarrow \bot) \rightarrow \bot) \rightarrow \varphi} \text{(Prop 3)}$$

  $\rightarrow$ **To be done as an exercise!**

- $$\frac{\varphi_1 \quad \varphi_1 \rightarrow \varphi_2}{\varphi_2} \text{(Modus Ponens)}$$

```
symbol mp : Π (φ1 φ2 : #Pattern),
  Prf φ1 → Prf (φ1 ⇒_ML φ2) → Prf φ2;
```

A MATCHING LOGIC proof encoded into DEDUKTI

$$\dfrac{\dfrac{}{\Gamma \vdash \varphi \to \alpha}\,(P1) \qquad \dfrac{\dfrac{}{\Gamma \vdash \varphi \to (\alpha \to \varphi)}\,(P1) \qquad \dfrac{}{\Gamma \vdash (\varphi \to (\alpha \to \varphi)) \to ((\varphi \to \alpha) \to \alpha)}\,(P2)}{\Gamma \vdash (\varphi \to \alpha) \to \alpha}\,(MP)}{\Gamma \vdash \alpha}\,(MP)$$

avec $\alpha \equiv \varphi \to \varphi$

```
symbol imp-identity : Π φ0, Prf (φ0 ⇒_ML φ0) :=
  λ φ0,
    mp (φ0 ⇒_ML (φ0 ⇒_ML φ0))
       (φ0 ⇒_ML φ0)
       (prop-1 φ0 φ0)
       (mp (φ0 ⇒_ML ((φ0 ⇒_ML φ0) ⇒_ML φ0))
           ((φ0 ⇒_ML (φ0 ⇒_ML φ0)) ⇒_ML (φ0 ⇒_ML φ0))
           (prop-1 φ0 (φ0 ⇒_ML φ0))
           (prop-2 φ0 (φ0 ⇒_ML φ0) φ0));
```

FOL reasoning

$$\frac{}{\varphi[y/x] \quad \rightarrow \quad \exists x.\varphi} \ (\exists\text{-Quantifier})$$

$$\frac{\varphi_1 \quad \rightarrow \quad \varphi_2 \qquad (\textit{when } x \notin FV(\varphi_2))}{(\exists x.\varphi_1) \quad \rightarrow \quad \varphi_2} \ (\exists\text{-Generalization})$$

FOL reasoning

$$\frac{}{\varphi[y/x] \quad \rightarrow \quad \exists x.\varphi} \ (\exists\text{-Quantifier})$$

$$\frac{\varphi_1 \quad \rightarrow \quad \varphi_2 \qquad (\textit{when } x \notin FV(\varphi_2))}{(\exists x.\varphi_1) \quad \rightarrow \quad \varphi_2} \ (\exists\text{-Generalization})$$

Problems:

- Substitution
- Checking of free variable

FOL reasoning

$$\frac{}{\varphi[y/x] \;\; \rightarrow \;\; \exists x.\varphi} \text{ ($\exists$-Quantifier)}$$

$$\frac{\varphi_1 \;\; \rightarrow \;\; \varphi_2 \qquad (\textit{when } x \notin FV(\varphi_2))}{(\exists x.\varphi_1) \;\; \rightarrow \;\; \varphi_2} \text{ ($\exists$-Generalization)}$$

Problems:

- Substitution
- Checking of free variable

$\rightarrow$ Solution: HOAS

FOL reasoning

- $$\frac{}{\varphi[y/x] \;\rightarrow\; \exists x.\varphi}\;(\exists\text{-Quantifier})$$

```
symbol ex-quantifier :
  Π(φ : #Element → #Pattern)
   (y : #Element),
  Prf (φ y ⇒_ML (∃_ML φ)) ;
```

Very close to $\frac{}{\varphi \;\rightarrow\; \exists x.\varphi}\;(\exists\text{-Quantifier})$
because α-renaming is done by the DEDUKTI binder.

FOL reasoning

- $$\frac{}{\varphi[y/x] \;\; \rightarrow \;\; \exists x.\varphi} \; (\exists\text{-Quantifier})$$

```
symbol ex-quantifier :
  Π(φ : #Element → #Pattern)
   (y : #Element),
  Prf (φ y ⇒ML (∃ML φ)) ;
```

- $$\frac{\varphi_1 \;\; \rightarrow \;\; \varphi_2 \qquad (\textit{when } x \notin FV(\varphi_2))}{(\exists x.\varphi_1) \;\; \rightarrow \;\; \varphi_2} \; (\exists\text{-Generalization})$$

```
symbol ex-generalization :
  Π (φ1 : #Element → #Pattern)
    (φ2 : #Pattern),
  (Π (x : #Element), Prf (φ1 x ⇒ML φ2) )
      → Prf ( (∃ML φ1) ⇒ML φ2 ) ;
```

Framing reasoning

$$\frac{}{C[\bot] \;\rightarrow\; \bot} \; (Propagation_{\bot})$$

$$\frac{}{C[\varphi_1 \;\vee\; \varphi_2] \;\rightarrow\; C[\varphi_1] \;\vee\; C[\varphi_2]} \; (Propagation_{\vee})$$

$$\frac{\varphi_1 \;\rightarrow\; \varphi_2}{C[\varphi_1] \;\rightarrow\; C[\varphi_2]} \; (\text{Framing})$$

$$\frac{(when\ x \notin FV(C))}{C[\exists x.\varphi] \;\rightarrow\; \exists x.C[\varphi]} \; (Propagation_{\exists})$$

Framing reasoning

$$\frac{}{C[\bot] \;\rightarrow\; \bot}\ (Propagation_{\bot})$$

$$\frac{}{C[\varphi_1 \;\vee\; \varphi_2] \;\rightarrow\; C[\varphi_1] \;\vee\; C[\varphi_2]}\ (Propagation_{\vee})$$

$$\frac{\varphi_1 \;\rightarrow\; \varphi_2}{C[\varphi_1] \;\rightarrow\; C[\varphi_2]}\ (\text{Framing})$$

$$\frac{(when\ x \notin FV(C))}{C[\exists x.\varphi] \;\rightarrow\; \exists x.C[\varphi]}\ (Propagation_{\exists})$$

Problem:

- Application context $C ::= \Box \mid C \,@\, \varphi \mid \varphi \,@\, C$

Frame reasoning - Application context

Model the BNF grammar $C ::= \square \mid C \mathbin{@} \varphi \mid \varphi \mathbin{@} C$:

```
symbol #AC : TYPE ;
symbol HOLE : #AC ;
symbol ACleft  : #AC → #Pattern → #AC ;
symbol ACright : #Pattern → #AC → #AC ;
```

Frame reasoning - Application context

Model the BNF grammar $C ::= \square \mid C \mathbin{@} \varphi \mid \varphi \mathbin{@} C$:

```
symbol #AC : TYPE ;
symbol HOLE : #AC ;
symbol ACleft  : #AC → #Pattern → #AC ;
symbol ACright : #Pattern → #AC → #AC ;
```

Translate an application context into a pattern:

```
symbol AC2P : #AC → #Pattern → #Pattern ;
rule AC2P HOLE $x ↪ $x ;
rule AC2P (ACleft  $C $P) $x ↪ (AC2P $C $x) @ML $P ;
rule AC2P (ACright $P $C) $x ↪ $P @ML (AC2P $C $x) ;
```

Frame reasoning - Application context

Model the BNF grammar $C ::= \Box \mid C \,@\, \varphi \mid \varphi \,@\, C$:

```
symbol #AC : TYPE ;
symbol HOLE : #AC ;
symbol ACleft  : #AC → #Pattern → #AC ;
symbol ACright : #Pattern → #AC → #AC ;
```

Translate an application context into a pattern:

```
symbol AC2P : #AC → #Pattern → #Pattern ;
rule AC2P HOLE $x ↪ $x ;
rule AC2P (ACleft  $C $P) $x ↪ (AC2P $C $x) @ML $P ;
rule AC2P (ACright $P $C) $x ↪ $P @ML (AC2P $C $x) ;
```

Translate the rule $\dfrac{}{C[\bot] \;\to\; \bot}\ (\mathit{Propagation}_\bot)$:

```
symbol propag-bot :
  Π(C : #AC), Prf (AC2P C ⊥ML ⇒ML ⊥ML);
```

```
type propag-bot (ACright (injSym •) HOLE) ;
  // Prf ((injSym •) @ML ⊥ML ⇒ML ⊥ML)
```

Frame reasoning - Rules

- $\dfrac{}{C[\bot] \;\rightarrow\; \bot}$ $(Propagation_{\bot})$ → **Already done!**
- $\dfrac{}{C[\varphi_1 \;\vee\; \varphi_2] \;\rightarrow\; C[\varphi_1] \;\vee\; C[\varphi_2]}$ $(Propagation_{\vee})$

  → **To be done as an exercise!**
- $\dfrac{\varphi_1 \;\rightarrow\; \varphi_2}{C[\varphi_1] \;\rightarrow\; C[\varphi_2]}$ (Framing) → **To be done as an exercise!**
- $\dfrac{(when\ x \notin FV(C))}{C[\exists x.\varphi] \;\rightarrow\; \exists x.C[\varphi]}$ $(Propagation_{\exists})$

  → **Combine HOAS + Application context**

Fixpoint reasoning

$$\frac{\varphi}{\varphi[\psi/X]} \text{ (Set Variable Substitution)}$$

$$\frac{}{\varphi[(\mu X.\varphi)/X] \rightarrow \mu X.\varphi} \text{ (PreFixpoint)}$$

$$\frac{\varphi[\psi/X] \rightarrow \psi}{(\mu X.\varphi) \rightarrow \psi} \text{ (Knaster-Tarski)}$$

Problem:

- Is there a problem?

Fixpoint reasoning

- $\dfrac{\varphi}{\varphi[\psi/X]}$ (Set Variable Substitution)
- $\dfrac{}{\varphi[(\mu X.\varphi)/X] \;\; \rightarrow \;\; \mu X.\varphi}$ (PreFixpoint)

```
symbol Pre-fixpoint :
  Π (φ : #Pattern → #Pattern),
  Prf ( φ (μ_ML φ) ⇒_ML (μ_ML φ) ) ;
```

- $\dfrac{\varphi[\psi/X] \;\; \rightarrow \;\; \psi}{(\mu X.\varphi) \;\; \rightarrow \;\; \psi}$ (Knaster-Tarski)

```
symbol Knaster-Tarski :
  Π(φ : #Pattern → #Pattern)
   (ψ : #Pattern),
  Prf ( φ ψ ⇒_ML ψ ) →
  Prf ( (μ_ML φ) ⇒_ML ψ ) ;
```

```
symbol μ_ML  : (#Pattern → #Pattern) → #Pattern;
```

Fixpoint reasoning

- $\dfrac{\varphi}{\varphi[\psi/X]}$ (Set Variable Substitution) $\rightarrow$ **X is a free variable!**
- $\dfrac{}{\varphi[(\mu X.\varphi)/X] \;\rightarrow\; \mu X.\varphi}$ (PreFixpoint)

```
symbol Pre-fixpoint :
  Π (φ : #Pattern → #Pattern),
  Prf ( φ (μ_ML φ) ⇒_ML (μ_ML φ) ) ;
```

- $\dfrac{\varphi[\psi/X] \;\rightarrow\; \psi}{(\mu X.\varphi) \;\rightarrow\; \psi}$ (Knaster-Tarski)

```
symbol Knaster-Tarski :
  Π(φ : #Pattern → #Pattern)
   (ψ : #Pattern),
  Prf ( φ ψ ⇒_ML ψ ) →
  Prf ( (μ_ML φ) ⇒_ML ψ ) ;
```

```
symbol μ_ML : (#Pattern → #Pattern) → #Pattern;
```

The last problem

- $\dfrac{\varphi}{\varphi[\psi/X]}$ (Set Variable Substitution)

```
symbol Set-var-subst :
  Π (φ ψ : #Pattern) (n : nat),
    Prf φ → Prf (subst φ ψ n) ;
```

where the free variable is modelled by:

```
symbol Free : nat → #Set;
```

and the substitution is modelled by:

```
symbol subst :
  #Pattern → #Pattern → nat → #Pattern; // φ[ψ/X]

rule subst (injEl $x) _ _ ↪ injEl $x;

rule subst (injSet (Free $m)) $ψ $n ↪
 ite (eq $m $n) $ψ (injSet (Free $m));
```

Technical rules

$$\frac{}{\exists x.x}\ \text{(Existence)}$$

$$\frac{}{\neg(C_1[x \quad \wedge \quad \varphi] \quad \wedge \quad C_2[x \quad \wedge \quad \neg\varphi])}\ \text{(Singleton)}$$

To be done as an exercise!

1. A shallow encoding to execute a program in Dedukti

2. A deep encoding to check proofs in Dedukti
   - Translate Matching Logic constructors, notations and symbols
   - Translate Matching Logic proof system

3. Conclusion

To remember

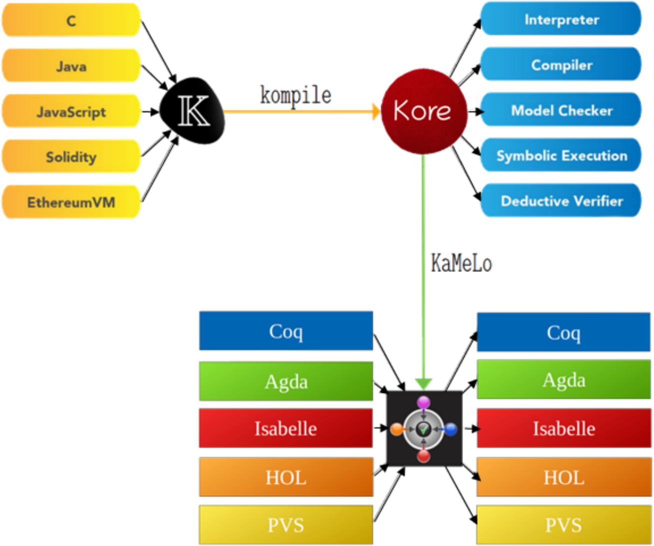
Computational part of an embedding

- Use rewriting rules!
  - Be careful about the expressivity of rewriting system!
  - Be careful to keep the confluence!
  - Be careful to keep the terminaison!

Axiomatic part of an embedding

- Model the provability relation: `symbol` Prf : #Pattern $\to$ `TYPE`
- Model variables and binders: HOAS vs De Bruijn indices
- Model grammar:
  - type as set
  - symbol as constructor

KaMeLo in action



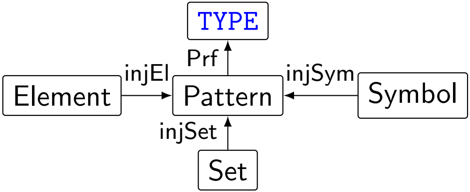
Typing

Matching Logic defines patterns φ.

- $\varphi ::= x \mid X \mid \sigma \mid \varphi\ \varphi \mid \bot \mid \varphi \rightarrow \varphi \mid \exists x.\varphi \mid \mu X.\varphi$

```
symbol @ML  : #Pattern → #Pattern → #Pattern;
notation @ML infix left 5;
symbol ⇒ML  : #Pattern → #Pattern → #Pattern;
symbol ∃ML  : (#Element → #Pattern) → #Pattern;
symbol μML  : (#Pattern → #Pattern) → #Pattern;
```

```
symbol #Pattern  : TYPE;
symbol #Element  : TYPE;
symbol #Set      : TYPE;
symbol #Symbol   : TYPE;
```




```
symbol injEl  : #Element → #Pattern;
symbol injSet : #Set → #Pattern;
symbol injSym : #Symbol → #Pattern;
```

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

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

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

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

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

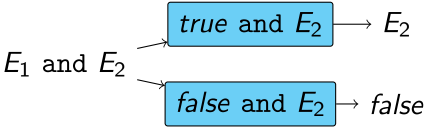
→ Everything is compiled in the same mechanism based on rewriting rules.

Evaluation strategy

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

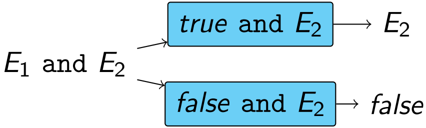
Evaluation strategy

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



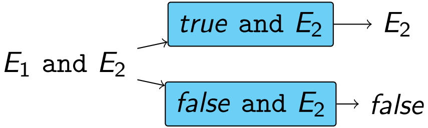
Evaluation strategy

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



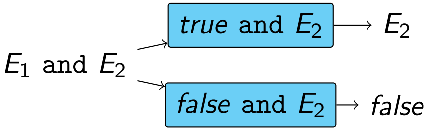
Evaluation strategy

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



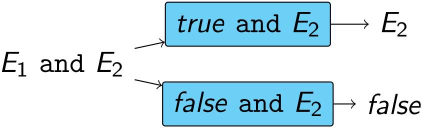
Evaluation strategy

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



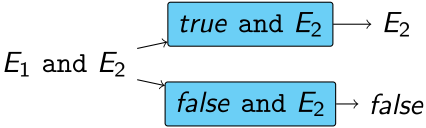
Evaluation strategy

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



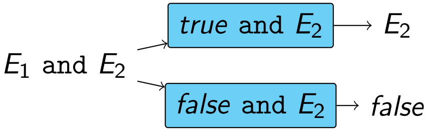
Evaluation strategy

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



Evaluation strategy

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



Evaluation strategy

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

  1. `rule` $\langle\ E_1 \text{ and } E_2 \curvearrowright s\ \rangle_k$m
     `=>` $\langle\ E_1 \curvearrowright (\circledast^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k$m `requires` $E_1 \notin$ `KResult`
  2. `rule` $\langle\ E_1 \curvearrowright (\circledast^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k$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`

Evaluation strategy

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

  $\langle\ (true \text{ and } false) \text{ and } (true \text{ and } true) \curvearrowright .\ \rangle_k$ .Map
  $\hookrightarrow_1 \langle\ (true \text{ and } false) \curvearrowright (\circledast^1_{\text{and}}\ (true \text{ and } true)) \curvearrowright .\ \rangle_k$ .Map

Evaluation strategy

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

  $\langle$ (*true* and *false*) and (*true* and *true*) $\curvearrowright$. $\rangle_k$ .Map
  $\hookrightarrow_1 \langle$ (*true* and *false*) $\curvearrowright (\circledast^1_{\text{and}}$ (*true* and *true*)) $\curvearrowright$. $\rangle_k$ .Map
  $\hookrightarrow_3 \langle$ *false* $\curvearrowright (\circledast^1_{\text{and}}$ (*true* and *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
     => $\langle\ E_1 \curvearrowright (❄^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k$m requires $E_1 \notin$ KResult
  2. rule $\langle\ E_1 \curvearrowright (❄^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k$m
     => $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k$m requires $E_1 \in$ KResult
  3. rule $\langle$ *true* and $b \curvearrowright s\ \rangle_k$m => $\langle\ b \curvearrowright s\ \rangle_k$m
  4. rule $\langle$ *false* and _ $\curvearrowright s\ \rangle_k$m => $\langle$ *false* $\curvearrowright s\ \rangle_k$m

  $\langle$ (*true* and *false*) and (*true* and *true*) $\curvearrowright$. $\rangle_k$ .Map
  $\hookrightarrow_1 \langle$ (*true* and *false*) $\curvearrowright (❄^1_{\text{and}}$ (*true* and *true*)) $\curvearrowright$. $\rangle_k$ .Map
  $\hookrightarrow_3 \langle$ *false* $\curvearrowright (❄^1_{\text{and}}$ (*true* and *true*)) $\curvearrowright$. $\rangle_k$ .Map
  $\hookrightarrow_2 \langle$ *false* and (*true* and *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
     => $\langle\ E_1 \curvearrowright (\circledast^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k$m requires $E_1 \notin$ KResult
  2. rule $\langle\ E_1 \curvearrowright (\circledast^1_{\text{and}}\ E_2) \curvearrowright s\ \rangle_k$m
     => $\langle\ E_1$ and $E_2 \curvearrowright s\ \rangle_k$m requires $E_1 \in$ KResult
  3. rule $\langle$ *true* and $b \curvearrowright s\ \rangle_k$m => $\langle\ b \curvearrowright s\ \rangle_k$m
  4. rule $\langle$ *false* and _ $\curvearrowright s\ \rangle_k$m => $\langle$ *false* $\curvearrowright s\ \rangle_k$m

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

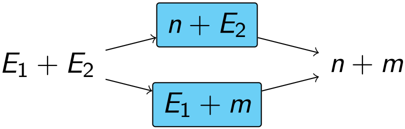
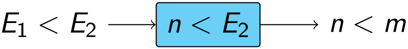
Evaluation strategy

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

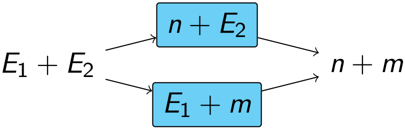
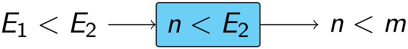
  $\langle$ (*true* and *false*) and (*true* and *true*) $\curvearrowright$. $\rangle_k$ .Map
  $\hookrightarrow_1 \langle$ (*true* and *false*) $\curvearrowright (\circledast^1_{\text{and}}$ (*true* and *true*)) $\curvearrowright$. $\rangle_k$ .Map
  $\hookrightarrow_3 \langle$ *false* $\curvearrowright (\circledast^1_{\text{and}}$ (*true* and *true*)) $\curvearrowright$. $\rangle_k$ .Map
  $\hookrightarrow_2 \langle$ *false* and (*true* and *true*) $\curvearrowright$. $\rangle_k$ .Map
  $\hookrightarrow_4 \langle$ *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`	Attribute `seqstrict`
	

Attribute strict

`rule` $E_1 + E_2$ `=>` $E_1 \curvearrowright (\text{❄}^1_{+}\ E_2)$
  `requires` $E_1 \notin$ `KResult`
`rule` $E_1 \curvearrowright (\text{❄}^1_{+}\ E_2)$ `=>` $E_1 + E_2$
  `requires` $E_1 \in$ `KResult`

`rule` $E_1 + E_2$ `=>` $E_2 \curvearrowright (\text{❄}^2_{+}\ E_1)$
  `requires` $E_2 \notin$ `KResult`
`rule` $E_2 \curvearrowright (\text{❄}^2_{+}\ E_1)$ `=>` $E_1 + E_2$
  `requires` $E_2 \in$ `KResult`

Attribute seqstrict

`rule` $E_1 < E_2$ `=>` $E_1 \curvearrowright (\text{❄}^1_{<}\ E_2)$
  `requires` $E_1 \notin$ `KResult`
`rule` $E_1 \curvearrowright (\text{❄}^1_{<}\ E_2)$ `=>` $E_1 < E_2$
  `requires` $E_1 \in$ `KResult`

`rule` $E_1 < E_2$ `=>` $E_2 \curvearrowright (\text{❄}^2_{<}\ E_1)$
  `requires` $E_2 \notin$ `KResult`
    $\wedge\ E_1 \in$ `KResult`
`rule` $E_2 \curvearrowright (\text{❄}^2_{<}\ E_1)$ `=>` $E_1 < E_2$
  `requires` $E_2 \in$ `KResult`

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

Overview of KProver

- Parametrized by a $\mathbb{K}$ semantic
- Based on the Reachability logic
  - → an extension of Hoare logic and separation logic
- Reachability property $\varphi \rightsquigarrow \varphi'$:

  During the execution of a program,
  if φ is **matched**, then φ' will be **matched** later on in a finite number of steps, or there is divergence.
- Example:

  $(N \geq 0) \wedge (S \geq 0) \wedge$

  $\langle\langle$ `while` $0 < \mathsf{n}$ do $\{ \mathsf{s} = \mathsf{s} + \mathsf{n}$; $\mathsf{n} = \mathsf{n}$ - 1; $\} \rangle_k \langle \mathsf{n} \mapsto \mathsf{N}, \mathsf{s} \mapsto \mathsf{S} \rangle_{env}\rangle$

  $\rightsquigarrow \langle\langle \ . \ \rangle_k \langle \mathsf{n} \mapsto 0, \mathsf{s} \mapsto \mathsf{S} + \frac{N*(N+1)}{2} \rangle_{env}\rangle$

Different rules for the same semantics

A. $\texttt{rule}\ \langle\langle\ x = i;\ \curvearrowright\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto \_)\ \rangle_{env}\rangle$
$\texttt{=>}\ \ \langle\langle\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto i)\ \rangle_{env}\rangle$

+ Implementation of Map in Dedukti (so without ACUI)

B. $\texttt{rule}\ \langle\langle\ x = i;\ \curvearrowright\ s\ \rangle_k\ \langle\ m\ \rangle_{env}\rangle$
$\texttt{=>}\ \ \langle\langle\ s\ \rangle_k\ \langle\ \text{update}\ m\ x\ i\ \rangle_{env}\rangle$

+ Implementation of Map in Dedukti (so without ACUI)

C. $\texttt{rule}\ \langle\langle\ x = i;\ \texttt{=>}\ \text{s}\ \ldots\ \rangle_k\ \langle\ \ldots\ (x \mapsto (\_\ \texttt{=>}\ i))\ \rangle_{env}\rangle$

+ Need to translate towards the rule B.[3]

[3] Work in progress of Everett Hildenbrandt.

Understand the problem

Is the rule:

$$\texttt{rule}\ \langle\langle\ x = i;\ \curvearrowright\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto \_)\ \rangle_{env}\rangle$$
$$\texttt{=>}\ \ \langle\langle\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto i)\ \rangle_{env}\rangle$$

applies in the following cases?

Understand the problem

Is the rule:

$\texttt{rule}\ \langle\langle\ x = i;\ \curvearrowright\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto \_)\ \rangle_{env}\rangle$
$\quad \texttt{=>}\ \ \langle\langle\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto i)\ \rangle_{env}\rangle$

applies in the following cases?

1. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (y \mapsto 52)\ \ (x \mapsto 42)\ \rangle_{env}\rangle$

Understand the problem

Is the rule:

$\texttt{rule}\ \langle\langle\ x = i;\ \curvearrowright\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto \_)\ \rangle_{env}\rangle$
$\texttt{=>}\ \ \langle\langle\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto i)\ \rangle_{env}\rangle$

applies in the following cases?

1. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (y \mapsto 52)\ \ (x \mapsto 42)\ \rangle_{env}\rangle$
2. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (x \mapsto 42)\ \ (y \mapsto 52)\ \rangle_{env}\rangle$

Understand the problem

Is the rule:

$\texttt{rule}\ \langle\langle\ x = i;\ \curvearrowright\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto \_)\ \rangle_{env}\rangle$
$\texttt{=>}\ \ \langle\langle\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto i)\ \rangle_{env}\rangle$

applies in the following cases?

1. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (y \mapsto 52)\ \ (x \mapsto 42)\ \rangle_{env}\rangle$
2. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (x \mapsto 42)\ \ (y \mapsto 52)\ \rangle_{env}\rangle$
3. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (x \mapsto 42)\ \rangle_{env}\rangle$

Understand the problem

Is the rule:

$$\texttt{rule}\ \langle\langle\ x = i; \curvearrowright s\ \rangle_k\ \langle\ m\ \ (x \mapsto \_)\ \rangle_{env}\rangle$$
$$\texttt{=>}\ \ \langle\langle\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto i)\ \rangle_{env}\rangle$$

applies in the following cases?

1. $\langle\langle\ x = 10; \curvearrowright\ .\ \rangle_k\ \langle\ (y \mapsto 52)\ \ (x \mapsto 42)\ \rangle_{env}\rangle$
2. $\langle\langle\ x = 10; \curvearrowright\ .\ \rangle_k\ \langle\ (x \mapsto 42)\ \ (y \mapsto 52)\ \rangle_{env}\rangle$

3. $\langle\langle\ x = 10; \curvearrowright\ .\ \rangle_k\ \langle\ (x \mapsto 42)\ \rangle_{env}\rangle$
4. $\langle\langle\ x = 10; \curvearrowright\ .\ \rangle_k\ \langle\ \ \texttt{.Map}\ \ (x \mapsto 42)\ \rangle_{env}\rangle$

Reminder: `.Map` = Empty environment

Understand the problem

Is the rule:

$$\texttt{rule}\ \langle\langle\ x = i;\ \curvearrowright\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto \_)\ \rangle_{env}\rangle$$
$$\texttt{=>}\ \ \langle\langle\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto i)\ \rangle_{env}\rangle$$

applies in the following cases?

1. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (y \mapsto 52)\ \ (x \mapsto 42)\ \rangle_{env}\rangle$
2. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (x \mapsto 42)\ \ (y \mapsto 52)\ \rangle_{env}\rangle$

3. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (x \mapsto 42)\ \rangle_{env}\rangle$
4. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ \ \texttt{.Map}\ \ (x \mapsto 42)\ \rangle_{env}\rangle$
5. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (x \mapsto 42)\ \ \texttt{.Map}\ \ \rangle_{env}\rangle$

Reminder: `.Map` = Empty environment

Understand the problem

Is the rule:

$\texttt{rule}\ \langle\langle\ x = i;\ \curvearrowright\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto \_)\ \rangle_{env}\rangle$

$\texttt{=>}\ \ \langle\langle\ s\ \rangle_k\ \langle\ m\ \ (x \mapsto i)\ \rangle_{env}\rangle$

applies in the following cases?

1. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (y \mapsto 52)\ \ (x \mapsto 42)\ \rangle_{env}\rangle$
2. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (x \mapsto 42)\ \ (y \mapsto 52)\ \rangle_{env}\rangle$

3. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (x \mapsto 42)\ \rangle_{env}\rangle$
4. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ \ \texttt{.Map}\ \ \ (x \mapsto 42)\ \rangle_{env}\rangle$
5. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ (x \mapsto 42)\ \ \ \texttt{.Map}\ \ \rangle_{env}\rangle$
6. $\langle\langle\ x = 10;\ \curvearrowright\ .\ \rangle_k\ \langle\ \ \texttt{.Map}\ \ \ (x \mapsto 42)\ \ \ \texttt{.Map}\ \ \ \ \texttt{.Map}\ \ \ \rangle_{env}\rangle$

Reminder: $\texttt{.Map}$ = Empty environment

Assessment & scaling up

- ✓ Syntax
- ✓ Configuration
- ✓ Evaluation strategy (context, attributes `strict` and `seqstrict`)
- ✓ Rewriting rule (conditional or not, attribute `owise`)

Assessment & scaling up

- ✓ Syntax
- ✓ Configuration
- ✓ Evaluation strategy (context, attributes `strict` and `seqstrict`)
- ✓ Rewriting rule (conditional or not, attribute `owise`)

- ✗ Rewriting modulo ACUI
- ✗ $\mathbb{K}$ standard library

Assessment & scaling up

Semantics	Last update	KORE file	DEDUKTI file	DEDUKTI check	Execution
Ethereum Virtual Machine (EVM)	2022	✓ (60k)	✓	✗	-
Michelson	2022	✓ (36k)	✓	✗	-
C	2022	✗	-	-	-
IELE	Dec 2021	-	-	-	-
WebAssembly	2022	✓ (30k)	✓	✗	-
Elrond	April 2021	✗	-	-	-
P4	May 2021	✓ (86k)	✓	✗	-
Plutus core	2022	✓ (40k)	✓	-	-
Java	Sept 2021	✗	-	-	-
Boogie	Sept 2021	✗	-	-	-
x86-64	2020	✗	-	-	-
Ewasm	2020	✗	-	-	-
Hybrid programs	2020	✗	-	-	-
K	2020	-	-	-	-
Yul	2019	✗	-	-	-
Ethereum Environment Interface (EEI)	2019	-	-	-	-
LLVM	2018	✗	-	-	-
Vyper	2018	-	-	-	-
ERC20	2018	-	-	-	-
ERC777	2018	-	-	-	-
Solidity	2017	-	-	-	-
Orc	2017	✗	-	-	-
Haskell core	2017	✗	-	-	-
Cink	2015	✗	-	-	-
JavaScript	2015	✗	-	-	-
JVM	2014	✗	-	-	-
JavaCard	2014	✗	-	-	-
Alk	2014	✗	-	-	-
AADL	2013	✗	-	-	-
Modelink	2013	✗	-	-	-
Python	2013	✗	-	-	-
OCaml	2013	✗	-	-	-