Simple Configuration

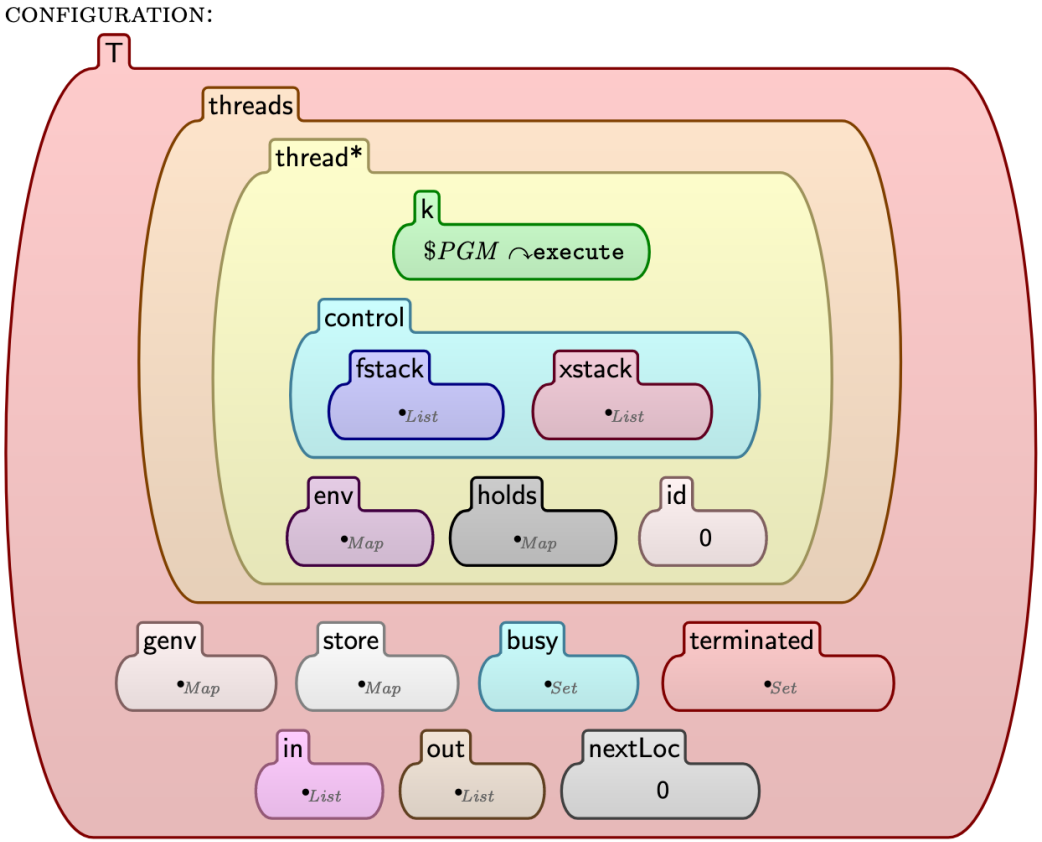
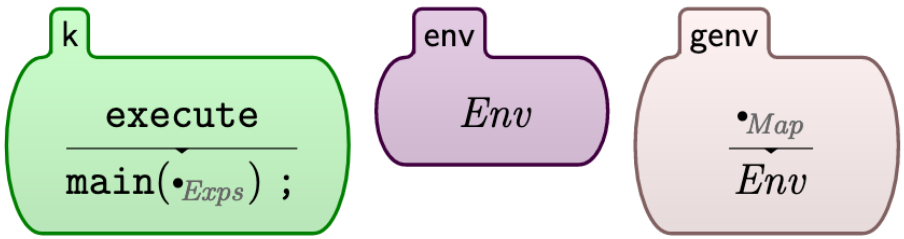


Fig. 1. The $\mathbb{K}$ configuration of SIMPLE

Starting execution

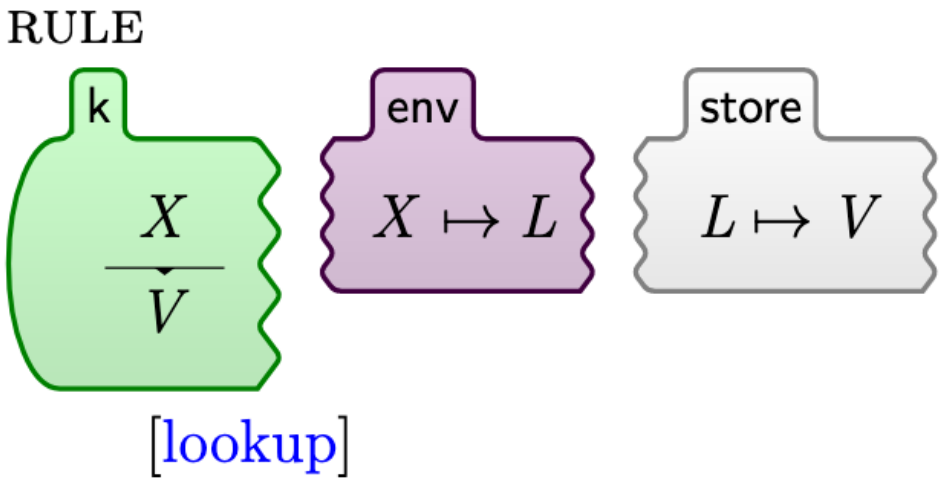
SYNTAX $K ::= \texttt{execute}$

RULE

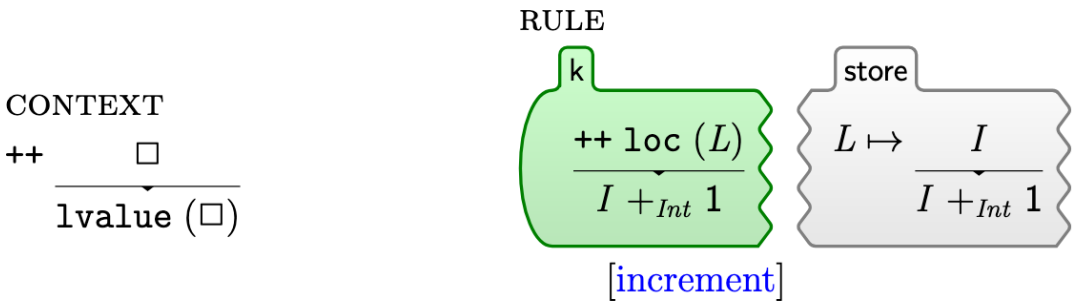


[structural]

Lookup

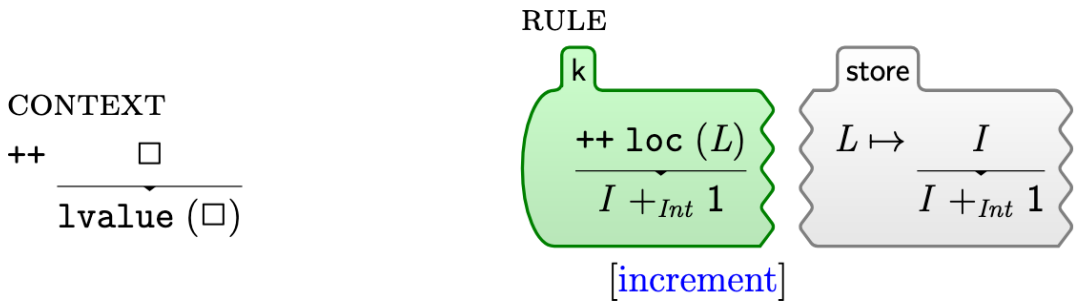


Increment



- lvalue described in the next slide

Increment



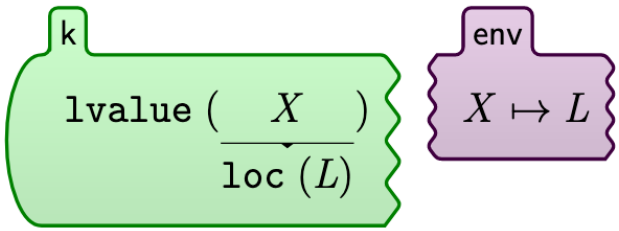
- lvalue described in the next slide
- Wrap *increment Exp* into an auxilliary lvalue construct.
- Once lvalue evaluates to *l*ocation loc(L), perform increment.

Increment (contd.)

SYNTAX $Exp ::= \texttt{lvalue}\ (K)$

SYNTAX $Val ::= \texttt{loc}\ (Int)$

RULE



[structural]

Arithemic Operators

RULE
$$\frac{I1 \texttt{ + } I2}{I1 +_{Int} I2}$$

RULE
$$\frac{Str1 \texttt{ + } Str2}{Str1 +_{String} Str2}$$

RULE
$$\frac{I1 \texttt{ - } I2}{I1 -_{Int} I2}$$

RULE
$$\frac{I1 \texttt{ * } I2}{I1 *_{Int} I2}$$

RULE
$$\frac{I1 \texttt{ / } I2}{I1 \div_{Int} I2} \quad \text{when } I2 \neq_I nt\ 0$$

RULE
$$\frac{I1 \texttt{ \% } I2}{I1 \%_{Int} I2} \quad \text{when } I2 \neq_I nt\ 0$$

RULE
$$\frac{\texttt{ - } I}{0 -_{Int} I}$$

RULE
$$\frac{I1 \texttt{ < } I2}{I1 <_{Int} I2}$$

RULE
$$\frac{I1 \texttt{ <= } I2}{I1 \leq_{Int} I2}$$

RULE
$$\frac{I1 \texttt{ > } I2}{I1 >_{Int} I2}$$

RULE
$$\frac{I1 \texttt{ >= } I2}{I1 \geq_{Int} I2}$$

RULE
$$\frac{V1 \texttt{ == } V2}{V1 =_K V2}$$

RULE
$$\frac{V1 \texttt{ != } V2}{V1 \neq_K V2}$$

Boolean Operators

$$\textsc{rule}\ \frac{!\ T}{\neg_{Bool}\, T} \qquad \textsc{rule}\ \frac{\texttt{true \&\&}\ E}{E} \qquad \textsc{rule}\ \frac{\texttt{false \&\&}\ \_}{\texttt{false}}$$

$$\textsc{rule}\ \frac{\texttt{true}\ ||\ \_}{\texttt{true}} \qquad \textsc{rule}\ \frac{\texttt{false}\ ||\ E}{E}$$

Array Lookup

RULE

$$\frac{V[N1\,,\,N2\,,\,Vs]}{V[N1][N2\,,\,Vs]}$$

[structural, anywhere]

RULE

$$\frac{\texttt{array}\ (L, \_)[N]}{\texttt{lookup}\ (L +_{Int} N)}$$

[structural, anywhere]

Array Lookup

RULE

$$\frac{V[N1\,,\,N2\,,\,Vs]}{V[N1][N2\,,\,Vs]}$$

[structural, anywhere]

RULE

$$\frac{\texttt{array}\ (L, \_)[N]}{\texttt{lookup}\ (L +_{Int} N)}$$

[structural, anywhere]

- Lookup in an array - how are expressions handled in array offset?

Array Lookup

RULE

$$\frac{V[N1\,,\,N2\,,\,Vs]}{V[N1][N2\,,\,Vs]}$$

[structural, anywhere]

RULE

$$\frac{\texttt{array}\ (L, \_)[N]}{\texttt{lookup}\ (L +_{Int} N)}$$

[structural, anywhere]

- Lookup in an array - how are expressions handled in array offset?
- what is anywhere?

Array Lookup

RULE

$$\frac{V[N1\,,\,N2\,,\,Vs]}{V[N1][N2\,,\,Vs]}$$

[structural, anywhere]

RULE

$$\frac{\texttt{array}\ (L, \_)[N]}{\texttt{lookup}\ (L +_{Int} N)}$$

[structural, anywhere]

- Lookup in an array - how are expressions handled in array offset?
- what is anywhere?

Array Lookup (contd.)

$$
\begin{aligned}
\textsc{syntax}\quad \mathit{Exp} ::= {} & \mathit{Int} \mid \mathit{Bool} \mid \mathit{String} \mid \mathit{Id} \\
\mid {} & (\mathit{Exp}) \ [\text{bracket}] \\
\mid {} & \texttt{++}\ \mathit{Exp} \\
\mid {} & \mathit{Exp}[\mathit{Exps}] \ [\text{strict}]
\end{aligned}
$$

Array Lookup (contd.)

$$\begin{aligned}\textsc{syntax}\quad Exp ::= {} & Int \mid Bool \mid String \mid Id \\ \mid {} & (Exp)\ [\text{bracket}] \\ \mid {} & \texttt{++}\ Exp \\ \mid {} & Exp[Exps]\ [\text{strict}]\end{aligned}$$

$$\textsc{syntax}\quad Exps ::= List\{Exp, \text{“,”}\}\ [\text{strict}]$$

Array Lookup (contd.)

$$\begin{aligned}\textsc{syntax}\quad Exp ::= {} & Int \mid Bool \mid String \mid Id \\ \mid {} & (Exp)\ [\text{bracket}] \\ \mid {} & \texttt{++}\ Exp \\ \mid {} & Exp[Exps]\ [\text{strict}]\end{aligned}$$

$$\textsc{syntax}\quad Exps ::= List\{Exp, \text{“,”}\}\ [\text{strict}]$$

$$\textsc{rule}\quad \texttt{lvalue}\ (\underline{\texttt{lookup}\ (L)})\ \texttt{loc}\ (L)\ [\text{structural}]$$

Array Size

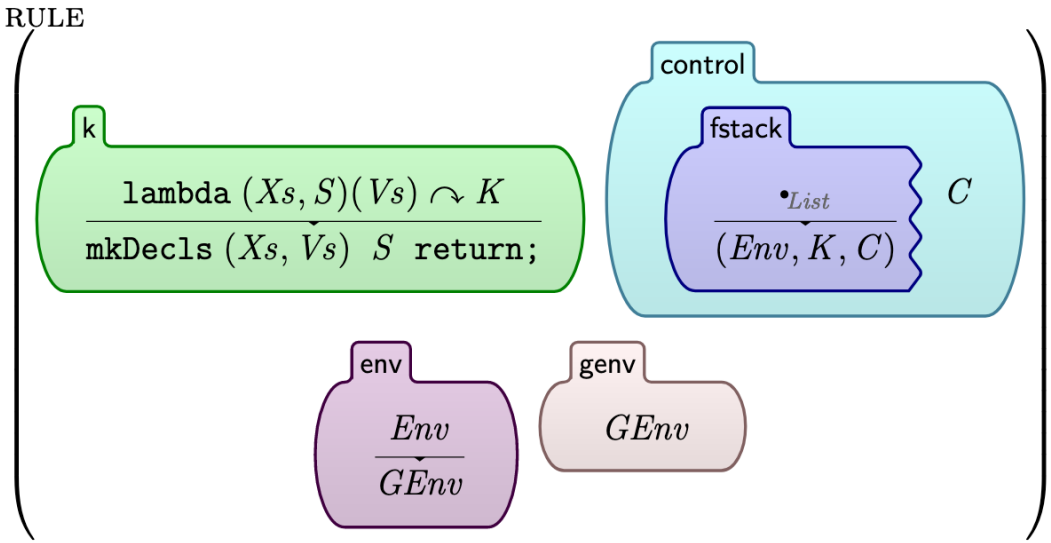
Size of an array

The size of the array is stored in the array reference value, and the `sizeOf` construct was declared strict, so:

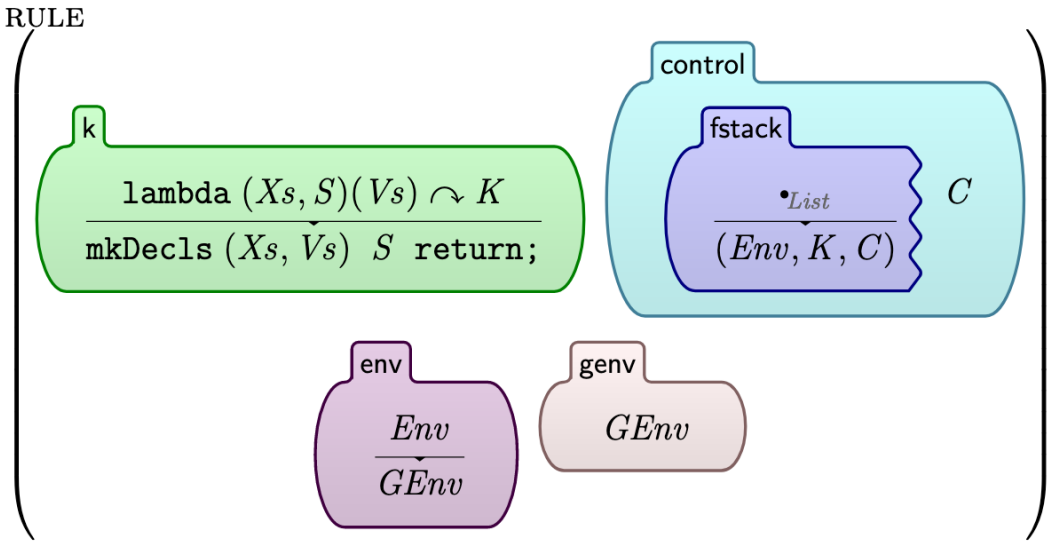
RULE

$$\frac{\texttt{sizeOf}\ (\texttt{array}\ (\text{—}, N))}{N}$$

Function Calls

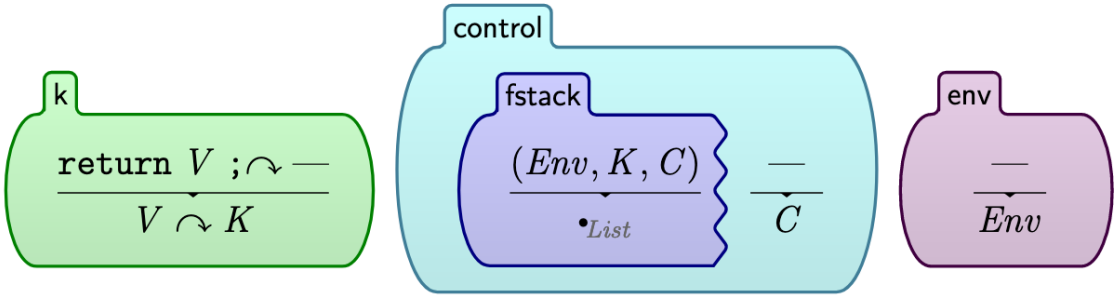


Function Calls



- Switch to *global environment*, where free variables in function are looked up.

Function Calls



SYNTAX $Val ::= \texttt{nothing}$

RULE

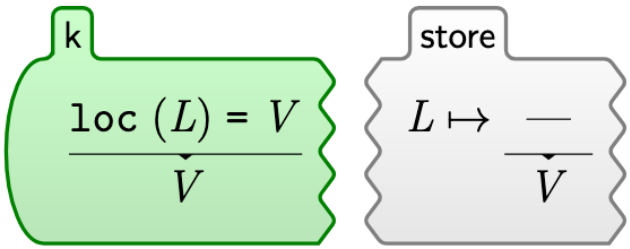
$$\frac{\texttt{return;}}{\texttt{return nothing ;}}\ [\text{macro}]$$

Assignments & Reads

CONTEXT

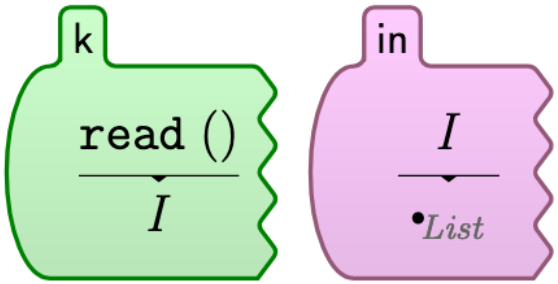
$$\frac{\square \quad = \ \_\_}{\texttt{lvalue}\ (\square)}$$

RULE



[assignment]

RULE



[read]

Sequential Composition, Expressions & Conditionals

RULE

$$\frac{S1 \quad S2}{S1 \curvearrowright S2}$$

[structural]

RULE

$$\frac{V \ ;}{\bullet K}$$

RULE

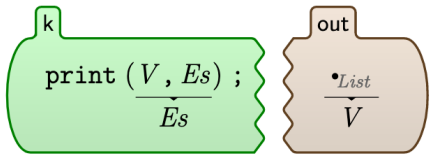
$$\frac{\texttt{if (true)}\, S\, \texttt{else} \ \_}{S}$$

RULE

$$\frac{\texttt{if (false)} \_\, \texttt{else}\ S}{S}$$

While & Print

RULE



[print]

RULE

$$\frac{\texttt{print}\ (\bullet_{Vals})\ ;}{\bullet_{K}}$$

[structural]

RULE

$$\frac{\texttt{while}\ (E)S}{\texttt{if}\ (E)\{S\ \ \texttt{while}\ (E)S\}}$$

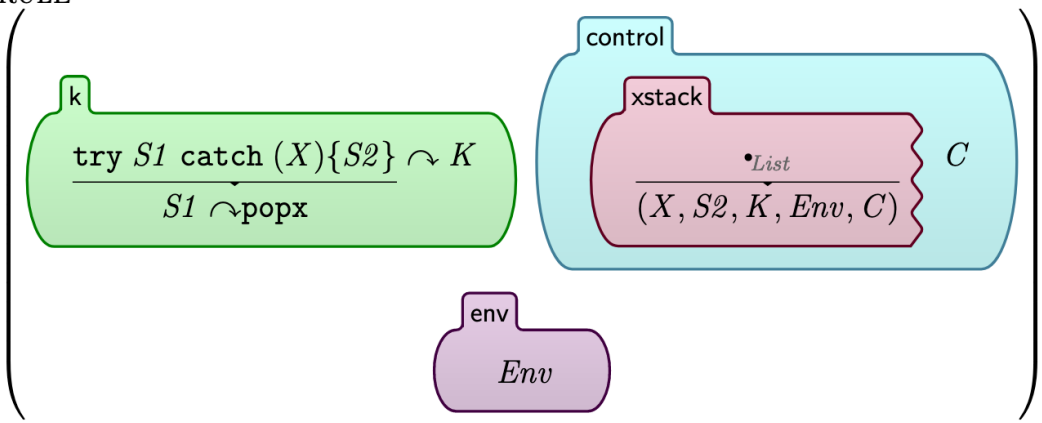
[structural]

Exception Handling

SYNTAX $ListItem ::= (Id, Stmt, K, Map, Bag)$

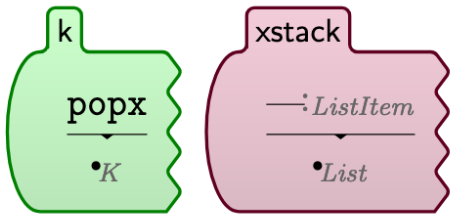
SYNTAX $K ::= \texttt{popx}$

RULE

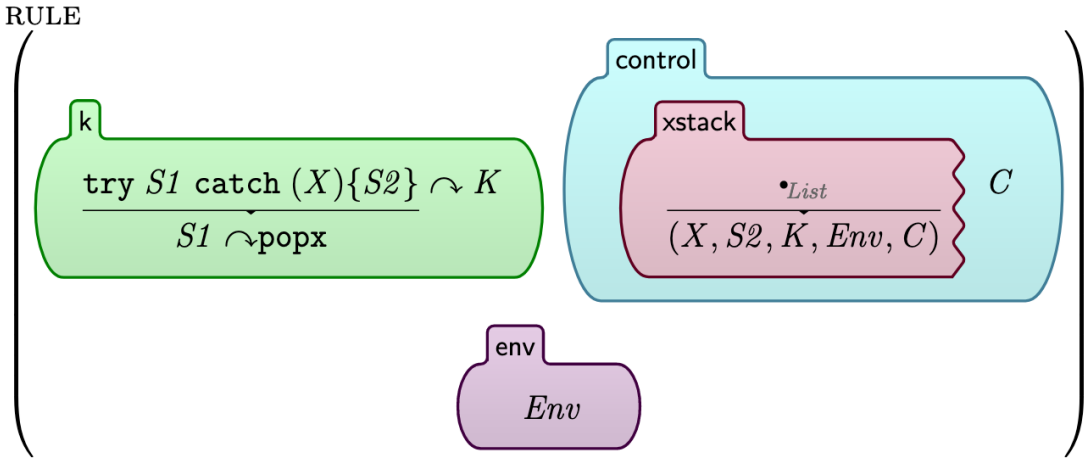


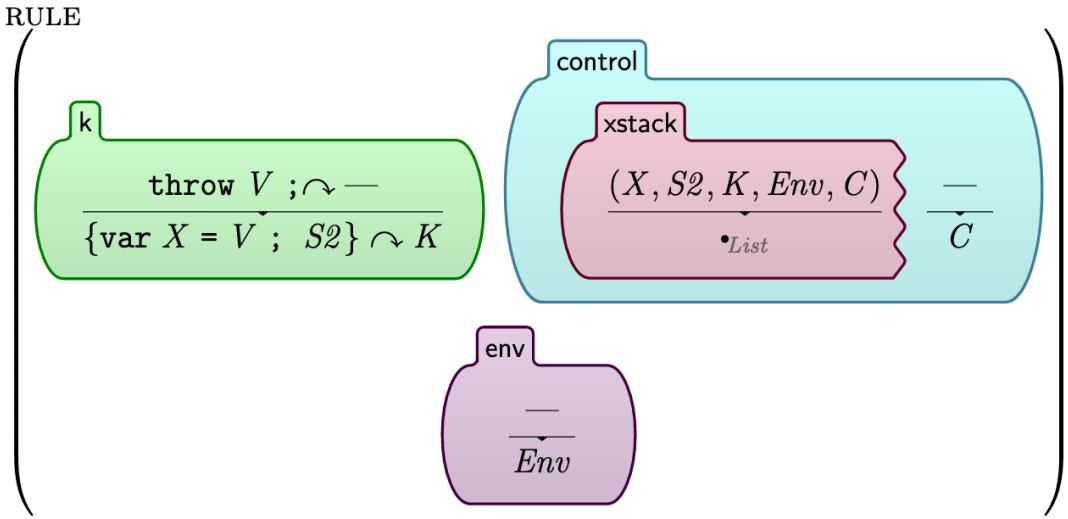
RULE

RULE



Exception Handling (contd.)

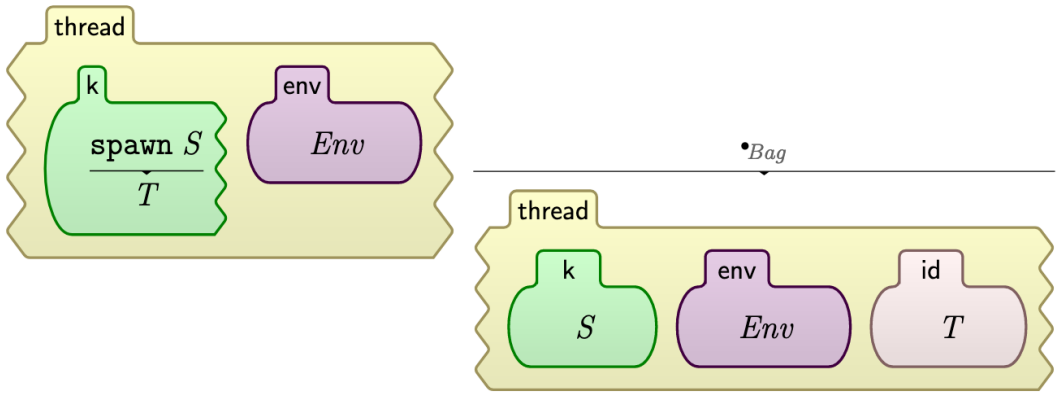




Concurrency

Spawn

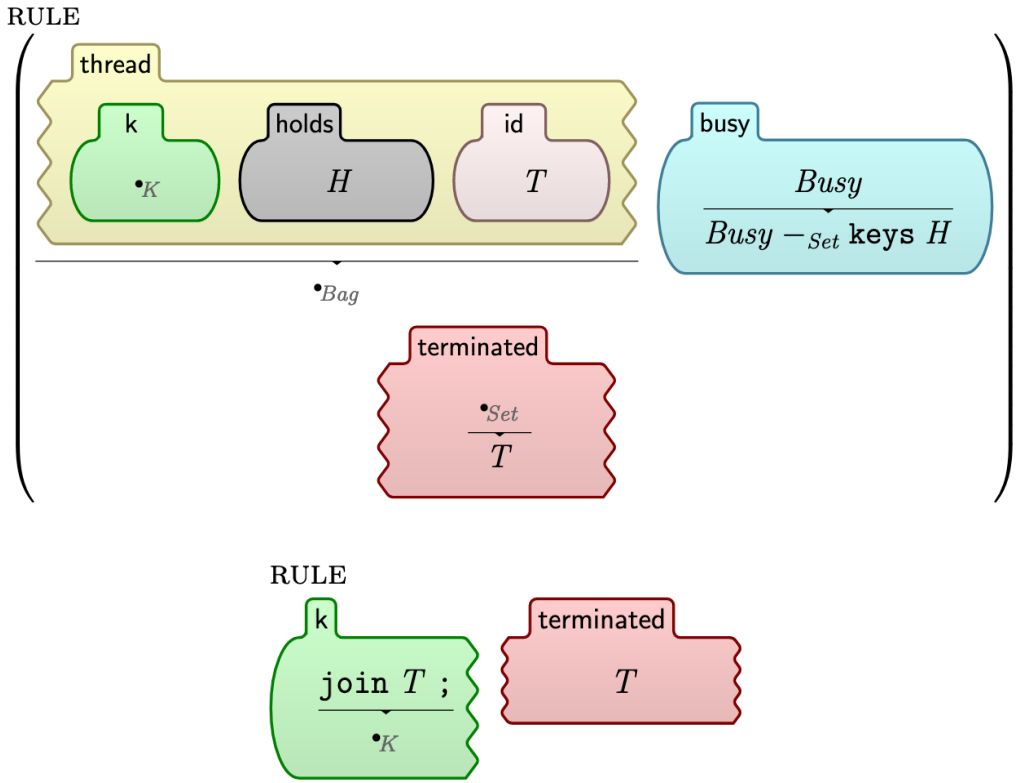
RULE



when `fresh` (T)

Concurrency

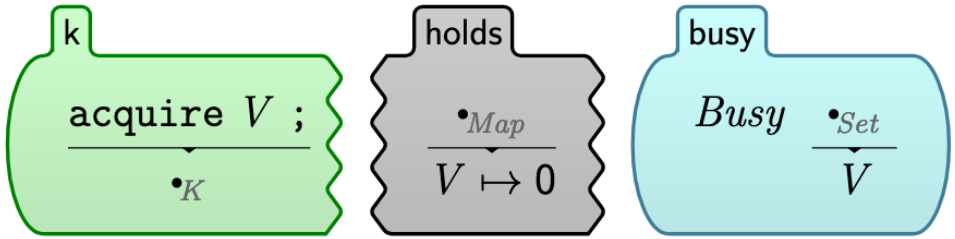
Termination & Joining



Concurrency

Acquire Locks

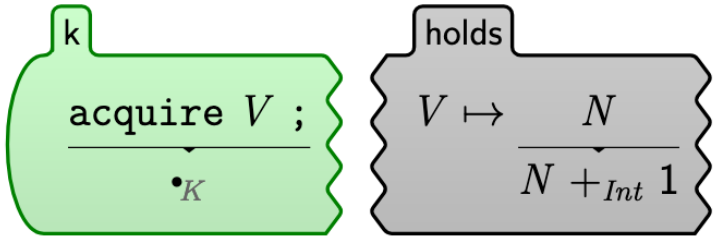
RULE



when $\neg_{Bool} V$ `in` $Busy$

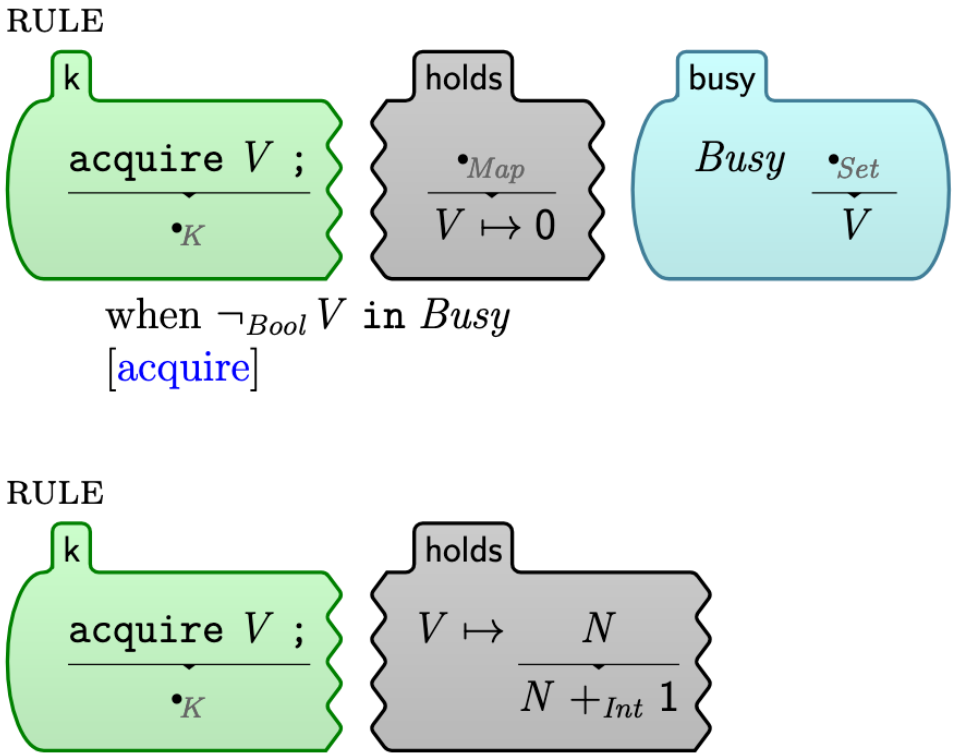
[acquire]

RULE



Concurrency

Acquire Locks

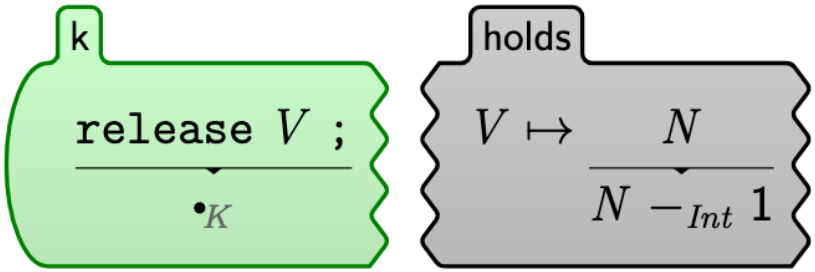


- Assume re-entrace. Same thread can acquire a lock again.

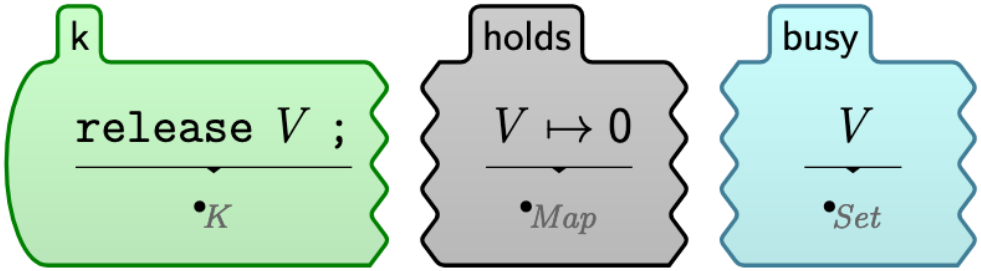
Concurrency

Release Locks

RULE

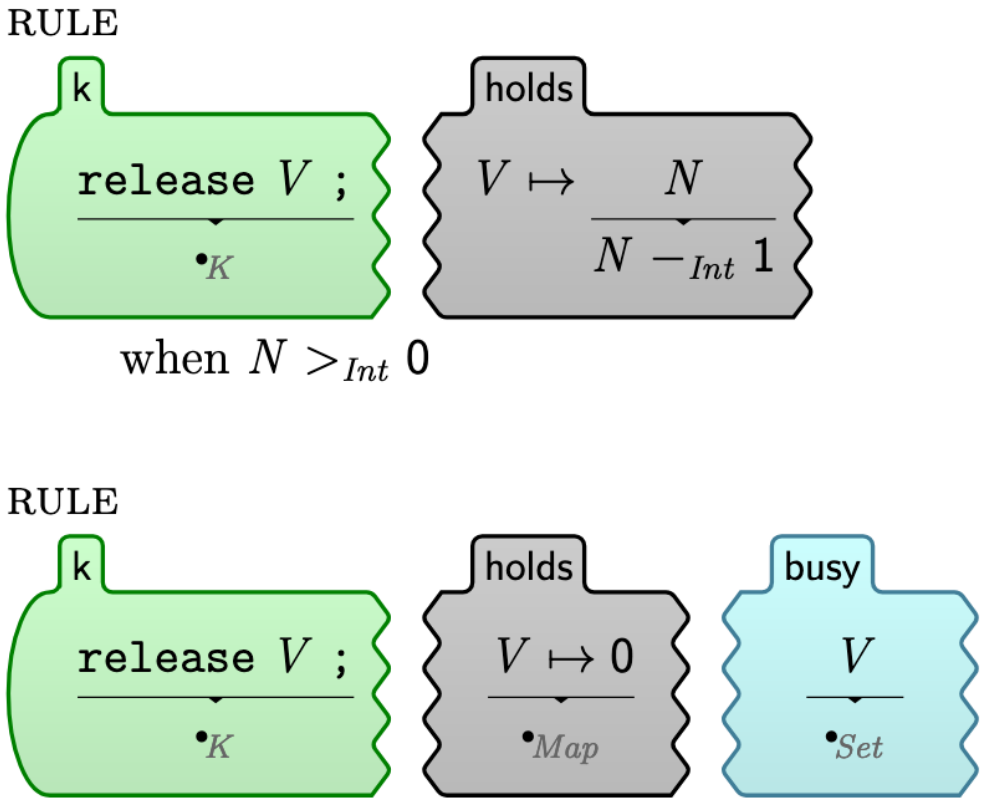


RULE



Concurrency

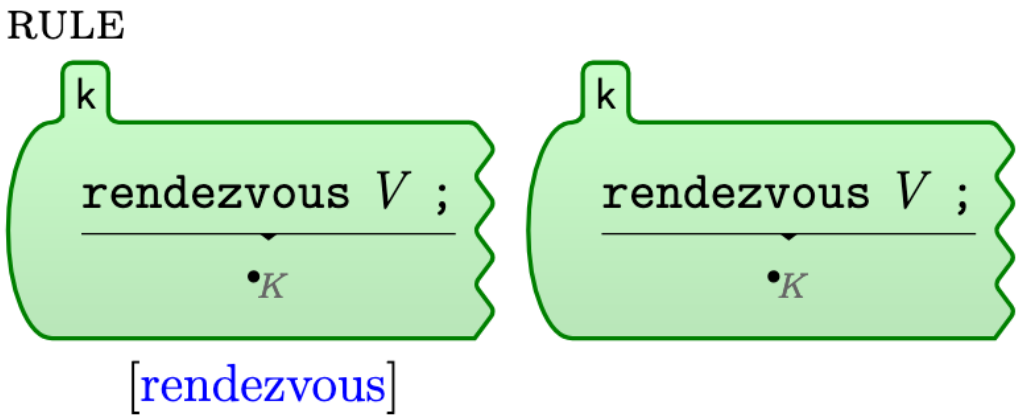
Release Locks



- A lock is considered release only when *n* release calls match with *n*-acquires calls.

Concurrency

Rendezvous



Auxilliary Constructs

Declarations, Lookups & Restoring Environments

SYNTAX $Decl ::= \texttt{mkDecls}\ (Ids, Vals)$ [function]

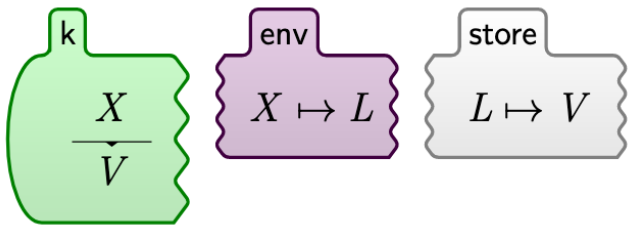
RULE

$$\frac{\texttt{mkDecls}\ ((X\ ,\ Xs), (V\ ,\ Vs))}{\texttt{var}\ X\ \texttt{=}\ V\ \texttt{;}\ \texttt{mkDecls}\ (Xs, Vs)}$$

RULE

$$\frac{\texttt{mkDecls}\ (\bullet_{Ids}, \bullet_{Vals})}{\{\}}$$

RULE



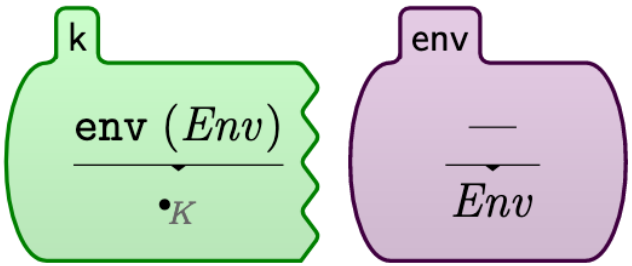
[lookup]

Auxilliary Constructs

Restoring Environments

SYNTAX $K ::= \texttt{env}\ (Map)$

RULE



[structural]