

# Time debits in nested thunks: a proof of Okasaki's banker's queue

**Glen Mével**, François Pottier, Jacques-Henri Jourdan

Inria Paris & LMF Paris-Saclay, France

Journée Inria–Nomadic Labs, June 1st 2022

# A purely functional queue

We can implement an immutable queue using two lists *front* and *rear*:

```
type 'α queue = 'α list × 'α list
```

```
let push (front, rear) x =  
  (front, x :: rear)           – insert into rear list
```

```
let pop (front, rear) =  
  match front with           – if front is non-empty..  
  | x :: front' → Some (x, (front', rear)) – ...pop its head  
  | [] →                   – otherwise..  
    match List.rev rear with – ...reverse rear to front (costly)...  
    | x :: front' → Some (x, (front', [])) – ...and pop head  
    | [] → None
```

The “banker’s method” (Tarjan, 1985) gives constant **amortized** costs:

- *push* costs  $\mathcal{O}(1)$ :
  - we spend  $\mathcal{O}(1)$  for cons-ing this element
  - we **save**  $\mathcal{O}(1)$ , covering for this element’s future reversal
- *pop* costs  $\mathcal{O}(1)$ :
  - we spend  $\mathcal{O}(1)$  for the call to *pop* itself
  - reversal is pre-paid by past *pushes*

**Issue:** we can't spend time savings twice

**let**  $q = \text{push}(\text{push}(\text{push nil } 1) 2) 3$  **in**

**let**  $(x_1, q_1) = \text{pop } q$  **in**    *– we spend our savings here*

**let**  $(x_2, q_2) = \text{pop } q$  **in**    *– wrong! we don't have any savings anymore*

...

⇒ Amortized complexity breaks if an old version of the queue is used

Idea (Okasaki, 1999):

- ① Compute reversals once ⇒ memoize them
- ② Share reversals among futures ⇒ suspend them ahead of time

⇒ **Laziness!**

The front sequence is a stream, i.e., a list computed on-demand:

```
type 'α stream = 'α cell thunk
and 'α cell = Nil | Cons of 'α × 'α stream
```

```
type 'α queue = int × 'α stream × int × 'α list
```

We enforce that  $|f| \geq |r|$ :

```
let rebalance ((lenf, f, lenr, r) as q) =
  assert (lenf + 1 ≥ lenr);
  if lenf ≥ lenr then q else   – re-establish inv. when r grows larger than f:
    (lenf + lenr, Stream.append f (Stream.rev_of_list r), 0, [])
    – ↑ create a thunk that will reverse r when forced
```

```
let push (lenf, f, lenr, r) x =
  rebalance (...)           – rebalance with element inserted into r
```

```
let pop (lenf, f, lenr, r) =
  match Stream.pop f with   – force the head thunk of f
  ... rebalance (...) ...   – rebalance with head removed from f
```

# Amortized complexity of the banker's queue

Reversing  $|r|$  elements is costly, but is done after  $|f| \geq |r|$  calls to *pop*

⇒ We can **anticipate** the cost of reversal on that of previous *pops*

⇒ Constant amortized costs:

- *rebalance* costs  $\mathcal{O}(1)$
- *push* costs  $\mathcal{O}(1)$
- *pop* costs  $\mathcal{O}(1)$

Key idea: time is a resource,  $\$n$  (“ $n$  time credits”) allow taking  $n$  steps

- The non-lazy queue saves **credit** for a yet unknown computation  
⇒ Not duplicable (cannot forge money)
- The banker's queue repays a **debit** for an already known computation  
⇒ Duplicable (can waste money)  
⇒ The banker's queue can be used **persistently**
  - Remark: the value is computed only once the debit is repaid

Building blocks:

- A **thunk** is a suspended computation, it holds a debit:

$$isThunk\ t\ m\ \varphi \quad (m \in \mathbb{N})$$

Ownership of a thunk is duplicable:

$$isThunk\ t\ m\ \varphi \multimap isThunk\ t\ m\ \varphi \star isThunk\ t\ m\ \varphi$$

- A **stream** is a chain of nested thunks, it holds a list of debits:

$$\begin{aligned}
 isStream\ s\ [m_1, \dots, m_n]\ [v_1, \dots, v_n] &\triangleq \\
 isThunk\ s\ m_1\ (\lambda c_1. \exists s_2. c_1 = Cons(v_1, s_2) \star & \\
 isThunk\ s_2\ m_2\ (\lambda c_2. \exists s_3. c_2 = Cons(v_2, s_3) \star & \\
 \vdots & \\
 isThunk\ s_{n+1}\ 0\ (\lambda c_{n+1}. c_{n+1} = Nil) \dots &))
 \end{aligned}$$

Ownership of a stream is duplicable



We can anticipate an inner think's debit:

$$\text{e.g. } \frac{\text{isThink } t_1 \ m_1 \ (\lambda t_2. \text{isThink } t_2 \ m_2 \ \varphi)}{\text{isThink } t_1 \ (m_1 + m) \ (\lambda t_2. \text{isThink } t_2 \ (m_2 - m) \ \varphi)}$$

⇒ We can anticipate debits in a stream:

$$\text{e.g. } \frac{\text{isStream } s \ [\overbrace{A, \dots, A}^{n \text{ times}}, (n+1)B, \overbrace{0, \dots, 0}^{n \text{ times}}] \ [f_1, \dots, f_n, r_{n+1}, \dots, r_1]}{\text{isStream } s \ [A+B, \dots, A+B, B, 0, \dots, 0] \ [f_1, \dots, f_n, r_{n+1}, \dots, r_1]}$$

This is needed in the proof of the banker's queue

Danielsson (2008) gives a dependent type system (in Agda) for specifying and verifying amortized costs of programs with thunks

- semi-formal guarantee
- no ghost operations: must insert them in code, manually
  - must conform to a strict discipline, must balance branches' costs, payment creates a thunk, in-depth payment needs special care...*
- ad-hoc type system, not a general-purpose program logic

Mével et al. (2019) extend Iris with time credits  $\Rightarrow$  Iris<sup>\$</sup>

Today's work: thunks, streams and the banker's queue (WIP) in Iris<sup>\$</sup>

This talk: thunks, streams

① Introduction

② Iris<sup>\$</sup> in a nutshell

③ Specification and proof, without anticipation

④ Anticipation

⑤ Anticipation

Iris extended with an assertion  $\$n$  ( $n \in \mathbb{N}$ ) satisfying a few laws:

$$\begin{aligned} & \vdash \$0 \\ \$ (m + n) & \equiv \$m \star \$n \end{aligned}$$

We can throw credits away, but not forge or duplicate them

Each execution step **consumes** \$1:

$$\text{e.g. } \{\$1 \star \ell \mapsto v\} !\ell \{\lambda v'. v' = v \star \ell \mapsto v\}$$

### Theorem (Soundness)

*If  $\{ \$n \} e \{ True \}$  is derivable in Iris<sup>\$</sup>, then program  $e$  is safe and terminates in at most  $n$  steps.*

```
type 'α thunk = 'α thunk_contents ref
```

```
and 'α thunk_contents =
```

```
| Future of (unit → 'α)
```

```
| Busy
```

```
| Done of 'α
```

```
let create f =
```

```
  ref (Future f)
```

```
let force t =
```

```
  match ! t with
```

```
  | Future f →
```

```
    if not (compare_and_set t (Future f) Busy) – forbid concurrent forcing
```

```
      then exit ();
```

```
      let v = f () in – evaluate the thunk...
```

```
      t := Done v; – ...and memoize the result
```

```
      v
```

```
  | Busy → exit ()
```

```
  | Done v → v – forbid reentrancy
```

$$\begin{array}{cc}
 \{\$K_{cr} \star (\$n \multimap wp f() \{ \square \varphi \})\} & \{\$K_{frc} \star isThink t 0 \varphi\} \\
 \text{create } f & \text{force } t \\
 \{\lambda t. isThink t n \varphi\} & \{\lambda v. \varphi v\}
 \end{array}$$

PERSIST

$\text{persistent}(isThink t m \varphi)$

OVERESTIMATE

$$\frac{isThink t m_1 \varphi \quad m_1 \leq m_2}{isThink t m_2 \varphi}$$

PAY

$$\frac{isThink t m \varphi \quad \$p}{\Rightarrow isThink t (m - p) \varphi}$$

ANTICIPATE

$$\frac{isThink t m \varphi \quad \forall v. \$n \star \varphi v \Rightarrow \square \psi v}{\Rightarrow isThink t (m + n) \psi}$$


$$\{ \$K_{cr} \star (\$n \multimap wp f() \{ \square \varphi \}) \} \quad \{ \$K_{frc} \star isThink t 0 \varphi \}$$

*create f*
*force t*

$$\{ \lambda t. isThink t n \varphi \} \quad \{ \lambda v. \varphi v \}$$

PERSIST

A thunk can be forced twice:  
its postcond must be duplicable



OVERESTIMATE

$$\frac{isThink t m_1 \varphi \quad m_1 \leq m_2}{isThink t m_2 \varphi}$$

$$\frac{isThink t m \varphi \quad \$p}{\Rightarrow isThink t (m - p) \varphi}$$

ANTICIPATE

$$\frac{isThink t m \varphi \quad \forall v. \$n \star \varphi v \Rightarrow \square \psi v}{\Rightarrow isThink t (m + n) \psi}$$



$$\{ \$K_{cr} \star (\$n \multimap wp f() \{ \square \varphi \}) \} \quad \{ \$K_{frc} \star isThink t 0 \varphi \}$$

*create f*
*force t*

$$\{ \lambda t. isThink t n \varphi \} \quad \{ \lambda v. \varphi v \}$$

PERSIST

A thunk can be forced twice:  
its postcond must be persistent

OVERESTIMATE

$$\frac{isThink t m_1 \varphi \quad m_1 \leq m_2}{isThink t m_2 \varphi}$$

$$\frac{isThink t m \varphi \quad \$p}{\Rightarrow isThink t (m - p) \varphi}$$

ANTICIPATE

$$\frac{isThink t m \varphi \quad \forall v. \$n \star \varphi v \Rightarrow \square \psi v}{\Rightarrow isThink t (m + n) \psi}$$

$$\begin{array}{cc}
 \{\$K_{cr} \star (\$n \multimap wp f() \{ \square \varphi \})\} & \{\$K_{frc} \star isThink t 0 \varphi\} \\
 \text{create } f & \text{force } t \\
 \{\lambda t. isThink t n \varphi\} & \{\lambda v. \varphi v\}
 \end{array}$$

PERSIST

$\text{persistent}(isThink t m \varphi)$

OVERESTIMATE

$$\frac{isThink t m_1 \varphi \quad m_1 \leq m_2}{isThink t m_2 \varphi}$$

PAY

$$\frac{isThink t m \varphi \quad \$p}{\Rightarrow isThink t (m - p) \varphi}$$

ANTICIPATE

$$\frac{isThink t m \varphi \quad \forall v. \$n \star \varphi v \Rightarrow \square \psi v}{\Rightarrow isThink t (m + n) \psi}$$

A stream is a thunk which computes an element (its head) and another thunk (its tail):

```
type 'α stream = 'α cell thunk
and 'α cell = Nil | Cons of 'α × 'α stream
```

A stream has a list of debits, one before each element:

$$\begin{aligned}
 \text{isStream } s \ [m_1, \dots, m_n] \ [v_1, \dots, v_n] &\triangleq \\
 \text{isThunk } s \ m_1 \ (\lambda c_1. \exists s_2. c_1 = \text{Cons}(v_1, s_2) \star \\
 \text{isThunk } s_2 \ m_2 \ (\lambda c_2. \exists s_3. c_2 = \text{Cons}(v_2, s_3) \star \\
 \dots \\
 \text{isThunk } s_{n+1} \ 0 \ (\lambda c_{n+1}. c_{n+1} = \text{Nil} \dots))
 \end{aligned}$$

## (Selected rules) Specification of streams

$$\{ \$K_{\text{ap}} \star \text{isStream } s [m_1, \dots, m_n] [v_1, \dots, v_n] \star \text{isStream } s' [m'_1, \dots, m'_{n'}] [v'_1, \dots, v'_{n'}] \}$$

*append s s'*

$$\{ \lambda t. \text{isStream } t [A + m_1, \dots, A + m_n, m'_1, \dots, m'_{n'}] [v_1, \dots, v_n, v'_1, \dots, v'_{n'}] \}$$

$$\{ \$K_{\text{rv}} \star \text{isList } \ell [v_1, \dots, v_n] \}$$

*rev\_of\_list ℓ*

$$\{ \lambda s. \text{isStream } s [B \cdot n, 0, \dots, 0] [v_n, \dots, v_1] \}$$

PAYSTREAM

$$\text{isStream } s [m_1, m_2, \dots, m_n] [v_1, \dots, v_n] \quad \$p$$

$$\Rightarrow \text{isStream } s [m_1 - p, m_2, \dots, m_n] [v_1, \dots, v_n]$$

ANTICIPATE+OVERESTIMATESTREAM

$$\text{isStream } s [m_1, \dots, m_n] [v_1, \dots, v_n] \quad \forall k. \sum_{i \leq k} m_i \leq \sum_{i \leq k} m'_i$$

$$\Rightarrow \text{isStream } s [m'_1, \dots, m'_n] [v_1, \dots, v_n]$$

The banker's queue needs anticipation of debits in streams...

ANTICIPATE+OVERESTIMATESTREAM

$$\frac{\text{isStream } s [m_1, \dots, m_n] [v_1, \dots, v_n] \quad \forall k. \sum_{i \leq k} m_i \leq \sum_{i \leq k} m'_i}{\Rightarrow \text{isStream } s [m'_1, \dots, m'_n] [v_1, \dots, v_n]}$$

...therefore in thunks:

ANTICIPATE

*isThunk*  $t$   $m$   $\varphi$

$$\frac{}{\Rightarrow \text{isThunk } t (m + n) (\$n * \varphi)}$$

The banker's queue needs anticipation of debits in streams...

ANTICIPATE+OVERESTIMATESTREAM

$$\frac{\text{isStream } s [m_1, \dots, m_n] [v_1, \dots, v_n] \quad \forall k. \sum_{i \leq k} m_i \leq \sum_{i \leq k} m'_i}{\Rightarrow \text{isStream } s [m'_1, \dots, m'_n] [v_1, \dots, v_n]}$$

...therefore in thunks:

ANTICIPATE

*isThunk*  $t$   $m$   $\varphi$

$$\frac{}{\Rightarrow \text{isThunk } t (m + n) (\$n * \varphi)}$$

nonsensical, thunk  
postconditions  
must be persistent

The banker's queue needs anticipation of debits in streams...

ANTICIPATE+OVERESTIMATESTREAM

$$\frac{\text{isStream } s [m_1, \dots, m_n] [v_1, \dots, v_n] \quad \forall k. \sum_{i \leq k} m_i \leq \sum_{i \leq k} m'_i}{\Rightarrow \text{isStream } s [m'_1, \dots, m'_n] [v_1, \dots, v_n]}$$

...therefore in thunks:

ANTICIPATE

$$\frac{\text{isThunk } t \ m \ \varphi \quad \forall v. \$n \star \varphi \ v \Rightarrow \square \psi \ v}{\Rightarrow \text{isThunk } t \ (m + n) \ \psi}$$

The banker's queue needs anticipation of debits in streams...

ANTICIPATE+OVERESTIMATESTREAM

$$\frac{isStream\ s\ [m_1, \dots, m_n]\ [v_1, \dots, v_n] \quad \forall k. \sum_{i \leq k} m_i \leq \sum_{i \leq k} m'_i}{\Rightarrow isStream\ s\ [m'_1, \dots, m'_n]\ [v_1, \dots, v_n]}$$

...therefore in thinks:

ANTICIPATE

$$\frac{isThink\ t\ m\ \varphi \quad \forall v. \$n \star \varphi\ v \Rightarrow \square \psi\ v}{\Rightarrow isThink\ t\ (m + n)\ \psi}$$

Example: from rules PAY and ANTICIPATE we can derive:

$$\frac{isThink\ t_1\ m_1\ (\lambda t_2. isThink\ t_2\ m_2\ \varphi) \quad \frac{\$n \star isThink\ t_2\ m_2\ \varphi}{\Rightarrow isThink\ t_2\ (m_2 - n)\ \varphi} \text{ (PAY)}}{\Rightarrow isThink\ t_1\ (m_1 + n)\ (\lambda t_2. isThink\ t_2\ (m_2 - n)\ \varphi)} \text{ (ANTICIPATE)}$$



Three library layers: thunks (proven), streams (proven), queues (WIP)

In this talk:

- **anticipation** of debit
  - we overlooked it at first
  - non-trivial proof: tree of debits, many invariants
- streams are chains of nested thunks

Not in this talk:

- **reentrancy** forbidden statically
  - non-atomic invariants  $\implies$  thunks have **namespaces**
  - avoid reentrant streams  $\implies$  streams have **generations** (internally)
- full proof of the banker's queue
- **ghost debits!** (WIP)

<https://gitlab.inria.fr/gmevel/iris-time-proofs>

- DANIELSSON, N. A. 2008. *Lightweight semiformal time complexity analysis for purely functional data structures*. In *Principles of Programming Languages (POPL)*.
- MÉVEL, G., JOURDAN, J.-H., ET POTTIER, F. 2019. *Time credits and time receipts in Iris*. In *European Symposium on Programming (ESOP)*. Lecture Notes in Computer Science, vol. 11423. Springer, 1–27.
- OKASAKI, C. 1999. *Purely Functional Data Structures*. Cambridge University Press.
- TARJAN, R. E. 1985. *Amortized computational complexity*. *SIAM Journal on Algebraic and Discrete Methods* 6, 2, 306–318.



$$\begin{array}{l} \text{CREATEDEBIT} \\ \$m \Rightarrow \square Q \\ \hline \Rightarrow \text{debit } m Q \end{array}$$

$$\begin{array}{l} \text{FORCEDEBIT} \\ \text{debit } 0 Q \\ \hline \Rightarrow \blacktriangleright Q \end{array}$$

$$\begin{array}{l} \text{PERSISTDEBIT} \\ \text{persistent}(\text{debit } m Q) \end{array}$$

$$\begin{array}{l} \text{OVERESTIMATEDEBIT} \\ \text{debit } m_1 Q \quad m_1 \leq m_2 \\ \hline \text{debit } m_2 Q \end{array}$$

$$\begin{array}{l} \text{PAYDEBIT} \\ \text{debit } m Q \quad \$p \\ \hline \Rightarrow \text{debit } (m - p) Q \end{array}$$

$$\begin{array}{l} \text{ANTICIPATEDEBIT} \\ \text{debit } m Q \quad \$n \star Q \Rightarrow \square Q' \\ \hline \Rightarrow \text{debit } (m + n) Q' \end{array}$$

(assuming a ghost name  $\gamma_t$  for each location  $t$ , by convenience)

$$thunkInv\ t\ \varphi \triangleq \exists n. [\bullet n]^{\gamma_t} \star \bigvee \left\{ \begin{array}{l} \exists f. t \mapsto Future\ f \star (\$n \text{ -* } wp\ f() \{ \square \varphi \}) \\ t \mapsto Busy \\ \exists v. t \mapsto Done\ v \star \square \varphi\ v \end{array} \right.$$

$$isThink\ t\ m\ \varphi \triangleq [\circ m]^{\gamma_t} \star \boxed{thunkInv\ t\ \varphi}$$

Ghost state in  $AUTH(\bar{N}, \min)$  reflects the remaining cost:

(assuming a ghost name  $\gamma_t$  for each location  $t$ , by convenience)

$$thinkInv\ t\ \varphi \triangleq \exists n. \boxed{\bullet n}^{\gamma_t} \star \bigvee \begin{cases} \exists f. t \mapsto Future\ f \star (\$n \multimap wp\ f()\ \{\square\varphi\}) \\ t \mapsto Busy \\ \exists v. t \mapsto Done\ v \star \square\varphi\ v \end{cases}$$

$$isThink\ t\ m\ \varphi \triangleq \boxed{\circ m}^{\gamma_t} \star \boxed{thinkInv\ t\ \varphi}$$

Ghost state in  $AUTH(\bar{N}, \min)$  reflects the remaining cost:

- $\boxed{\bullet n}^{\gamma}$  asserts that the remaining cost is exactly  $n$  credits

(assuming a ghost name  $\gamma_t$  for each location  $t$ , by convenience)

$$thinkInv\ t\ \varphi \triangleq \exists n. \boxed{\bullet n}^{\gamma_t} \star \bigvee \begin{cases} \exists f. t \mapsto Future\ f \star (\$n \multimap wp\ f() \{\square\varphi\}) \\ t \mapsto Busy \\ \exists v. t \mapsto Done\ v \star \square\varphi\ v \end{cases}$$

$$isThink\ t\ m\ \varphi \triangleq \boxed{\circ m}^{\gamma_t} \star \boxed{thinkInv\ t\ \varphi}$$

Ghost state in  $AUTH(\bar{\mathbb{N}}, \min)$  reflects the remaining cost:


- $\boxed{\bullet n}^{\gamma}$  asserts that the remaining cost is exactly  $n$  credits
- $\boxed{\circ m}^{\gamma}$  witnesses that the remaining cost is at most  $m$  credits

(assuming a ghost name  $\gamma_t$  for each location  $t$ , by convenience)

$$thinkInv\ t\ \varphi \triangleq \exists n. \boxed{\bullet n}^{\gamma_t} \star \bigvee \left\{ \begin{array}{l} \exists f. t \mapsto Future\ f \star (\$n \multimap wp\ f() \{\square\varphi\}) \\ t \mapsto Busy \\ \exists v. t \mapsto Done\ v \star \square\varphi\ v \end{array} \right.$$

$$isThink\ t\ m\ \varphi \triangleq \boxed{\circ m}^{\gamma_t} \star \boxed{thinkInv\ t\ \varphi}$$

Ghost state in  $AUTH(\bar{\mathbb{N}}, \min)$  reflects the remaining cost:

- $\boxed{\bullet n}^{\gamma}$  asserts that the remaining cost is exactly  $n$  credits
- $\boxed{\circ m}^{\gamma}$  witnesses that the remaining cost is **at most**  $m$  credits  
 $\implies$  persistent 





(assuming a ghost name  $\gamma_t$  for each location  $t$ , by convenience)


$$\mathit{thinkInv} \ t \ \varphi \triangleq \exists n. \boxed{\bullet n}^{\gamma_t} \star \bigvee \left\{ \begin{array}{l} \exists f. t \mapsto \mathit{Future} \ f \ \star (\$n \multimap \mathit{wp} \ f() \ \{\square \varphi\}) \\ t \mapsto \mathit{Busy} \\ \exists v. t \mapsto \mathit{Done} \ v \ \star \square \varphi \ v \end{array} \right.$$

$$\mathit{isThink} \ t \ m \ \varphi \triangleq \boxed{\circ m}^{\gamma_t} \star \boxed{\mathit{thinkInv} \ t \ \varphi}$$

Ghost state in  $\text{AUTH}(\bar{\mathbb{N}}, \text{min})$  reflects the remaining cost:

- $\boxed{\bullet n}^{\gamma}$  asserts that the remaining cost is exactly  $n$  credits
- $\boxed{\circ m}^{\gamma}$  witnesses that the remaining cost is **at most**  $m$  credits  
 $\implies$  persistent 

OVERESTIMATE:  $\boxed{\circ m_1}^{\gamma} \multimap \boxed{\circ m_2}^{\gamma}$  if  $m_1 \leq m_2$  

PAY:  $\boxed{\bullet n}^{\gamma} \implies \boxed{\bullet (n-p)}^{\gamma} \star \boxed{\circ (n-p)}^{\gamma}$  

(assuming a ghost name  $\gamma_t$  for each location  $t$ , by convenience)

$$\mathit{thinkInv} \ t \ \varphi \triangleq \exists n. \boxed{\bullet n}^{\gamma_t} \star \bigvee \left\{ \begin{array}{l} \exists f. t \mapsto \mathit{Future} \ f \ \star (\$n \multimap \mathit{wp} \ f() \ \{\square \varphi\}) \\ t \mapsto \mathit{Busy} \\ \exists v. t \mapsto \mathit{Done} \ v \ \star \square \varphi \ v \end{array} \right.$$

$$\mathit{isThink} \ t \ m \ \varphi \triangleq \boxed{\circ m}^{\gamma_t} \star \boxed{\mathit{thinkInv} \ t \ \varphi}$$

Ghost state in  $\text{AUTH}(\bar{\mathbb{N}}, \text{min})$  reflects the remaining cost:

- $\boxed{\bullet n}^{\gamma}$  asserts that the remaining cost is exactly  $n$  credits
- $\boxed{\circ m}^{\gamma}$  witnesses that the remaining cost is **at most**  $m$  credits  
 $\implies$  persistent ✓

$$\text{OVERESTIMATE: } \boxed{\circ m_1}^{\gamma} \multimap \boxed{\circ m_2}^{\gamma} \text{ if } m_1 \leq m_2 \quad \checkmark$$

$$\text{PAY: } \boxed{\bullet n}^{\gamma} \implies \boxed{\bullet (n-p)}^{\gamma} \star \boxed{\circ (n-p)}^{\gamma} \quad \checkmark$$

spec of *create*: ✓spec of *force*:  $(m = 0) \implies (n = 0) \implies (\$n \equiv \text{emp}) \quad \checkmark$

Problems:

- known upper bounds  $\boxed{\circ m}$  must remain valid  $\implies$  can't increase  $\boxed{\bullet n}$
- $\varphi$  is fixed in the invariant  $\implies$  can't change it

Solution: stack a new debit, with a new invariant, on top of the old one!

Example scenario:

create a thunk with debit 5  
and postcondition  $A$

*isThunk t 5 A*

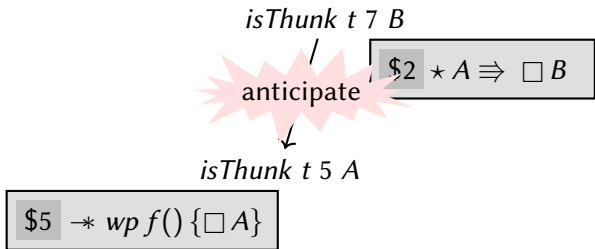
$\$5 \text{ } \rightarrow * \text{ wp } f() \{ \square A \}$

Example scenario:

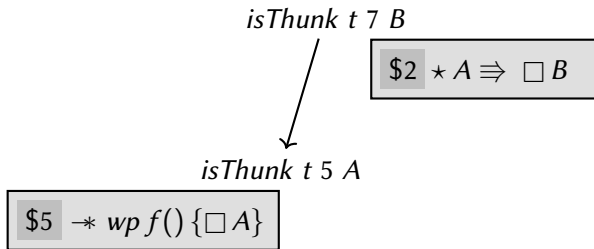
*isThunk t 5 A*

$\$5 \rightarrow * wp f() \{\square A\}$

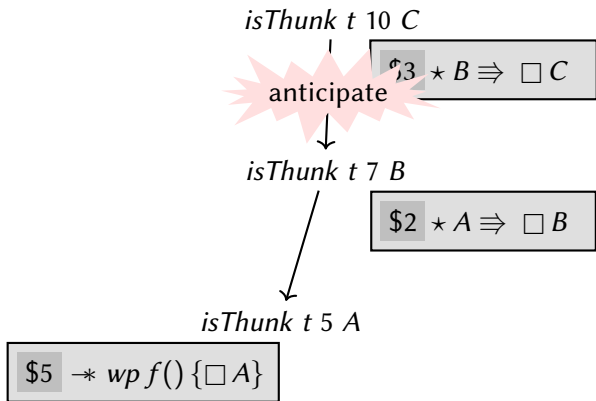
Example scenario:



Example scenario:

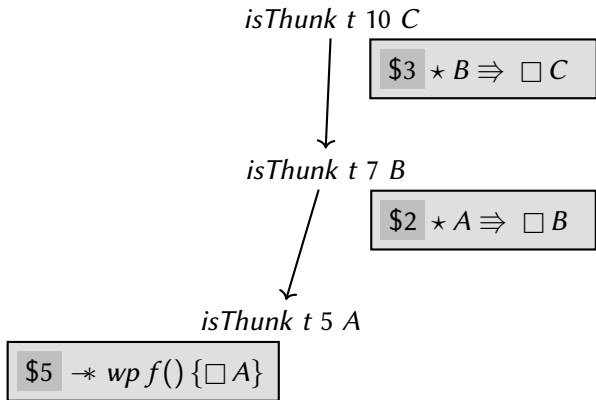


Example scenario:

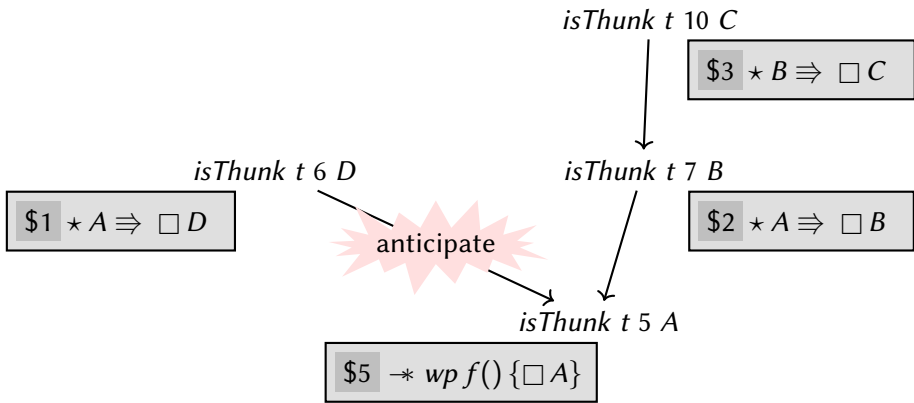




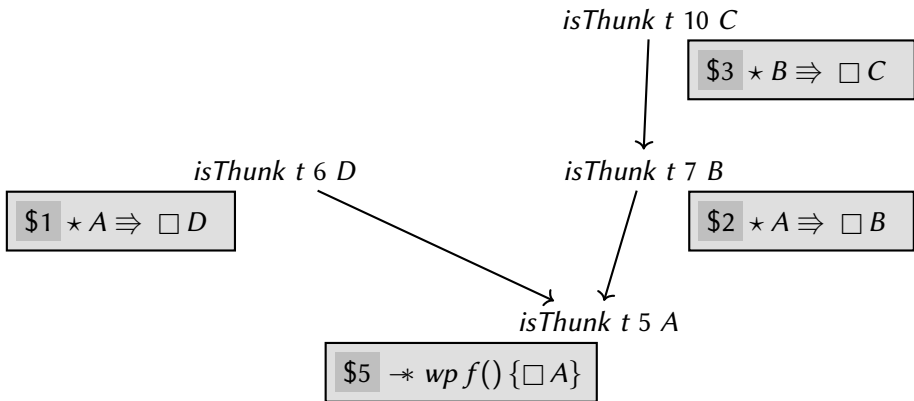
Example scenario:



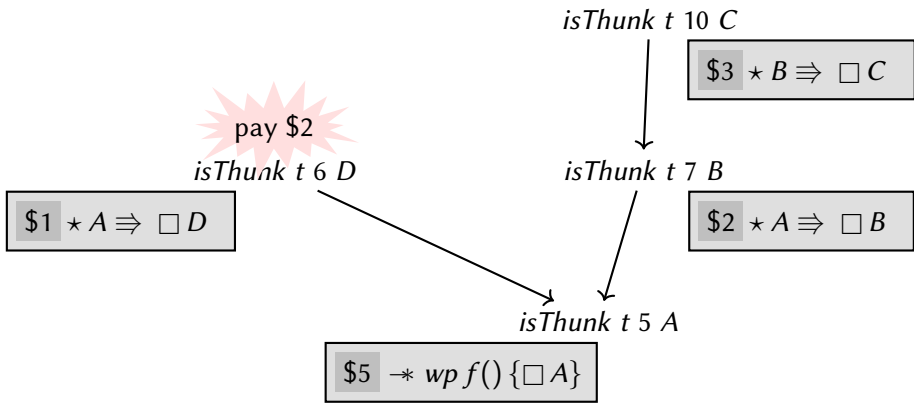
Example scenario:



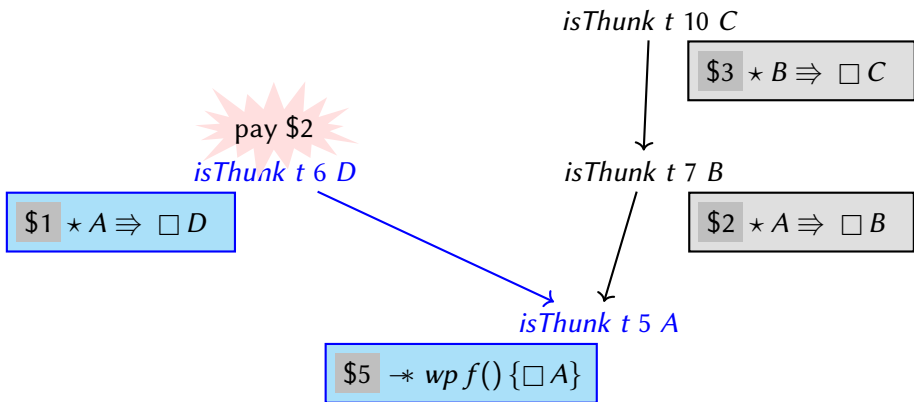
Example scenario:



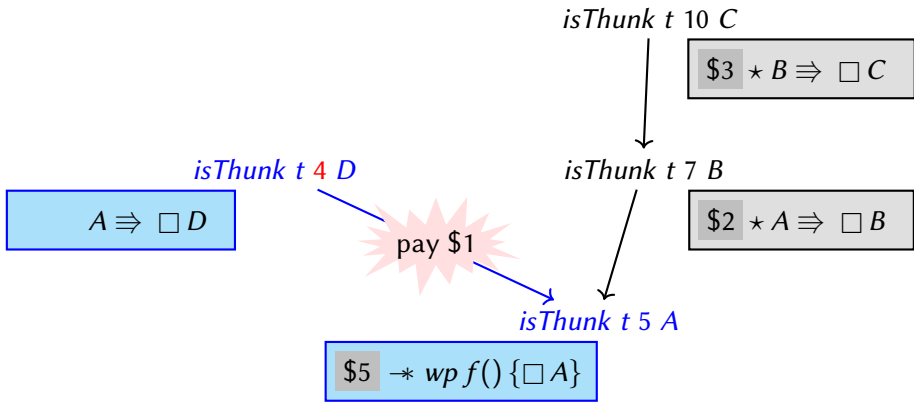
Example scenario:



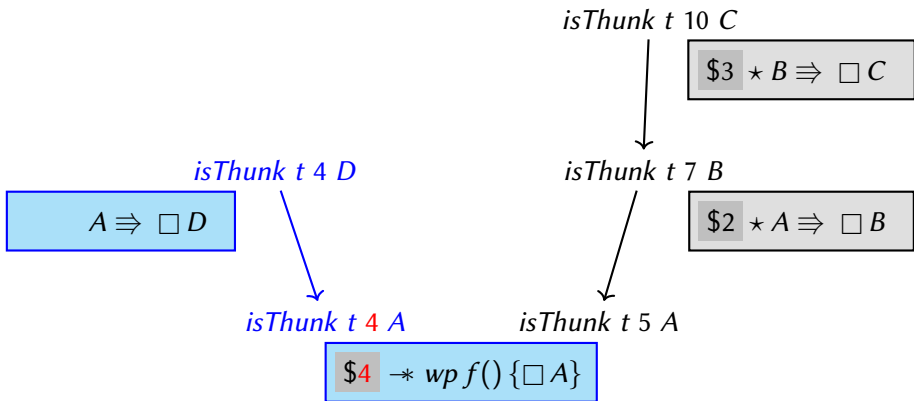
Example scenario:



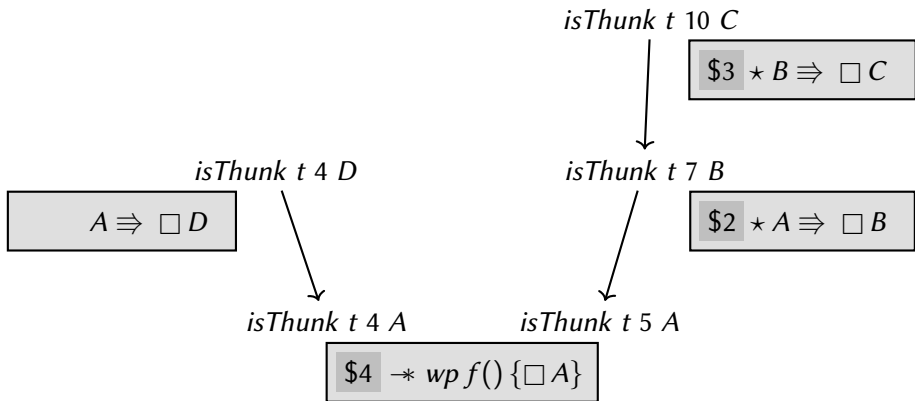
Example scenario:



Example scenario:

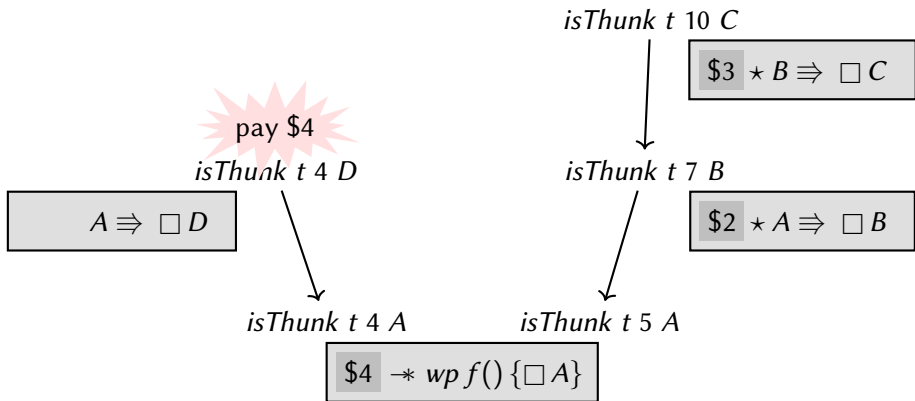


Example scenario:

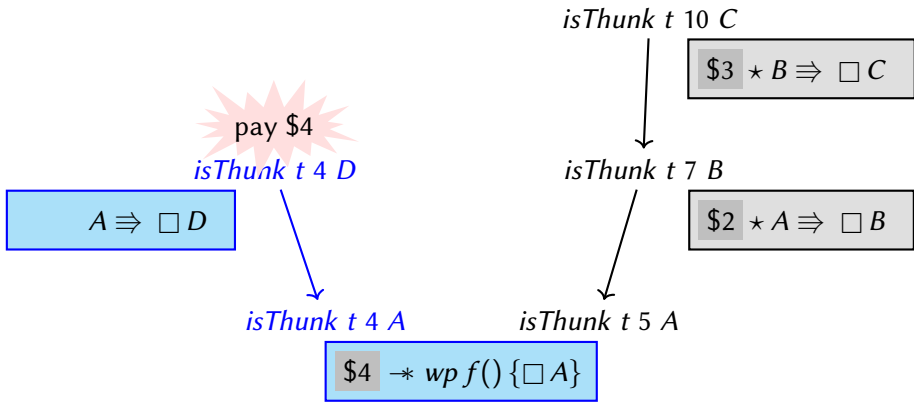




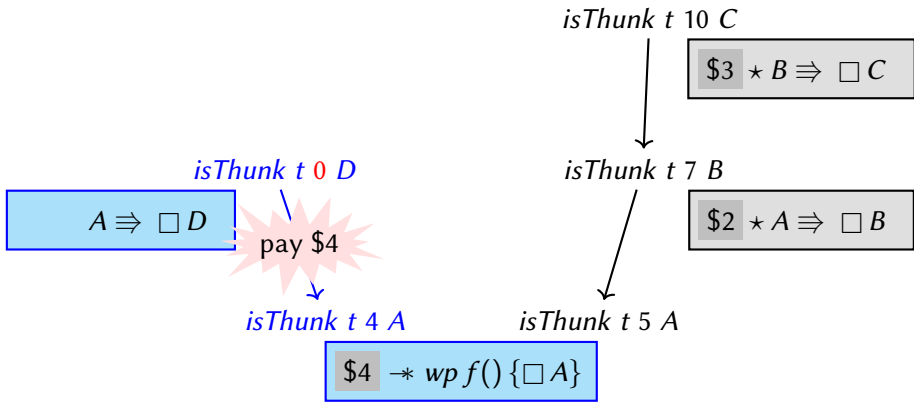
Example scenario:



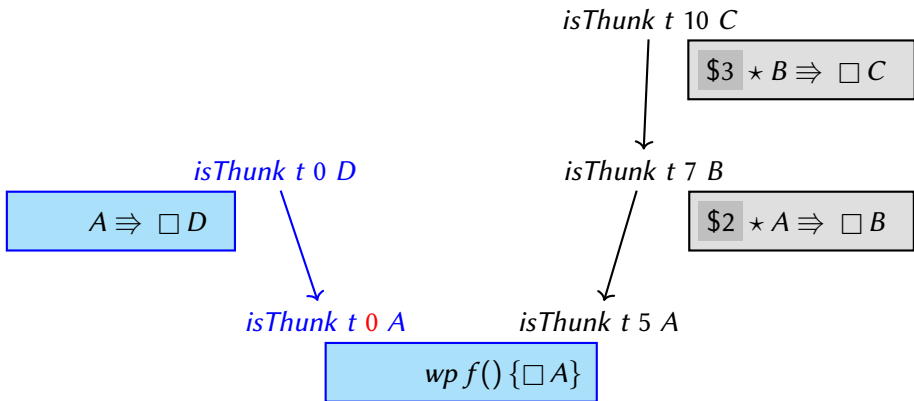
Example scenario:



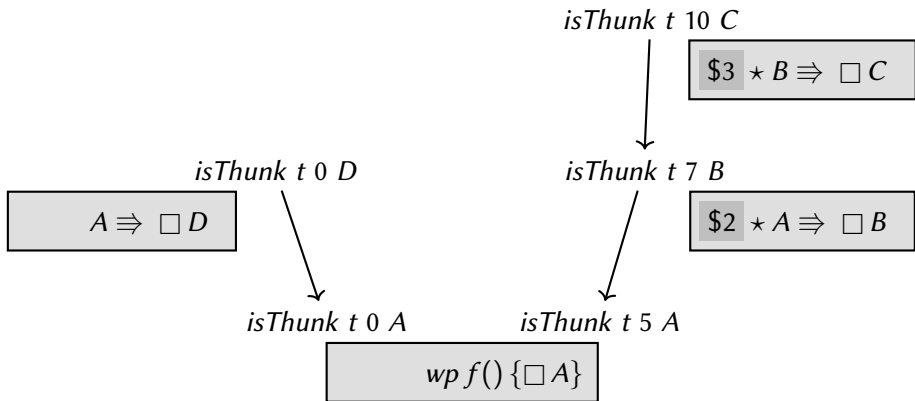
Example scenario:



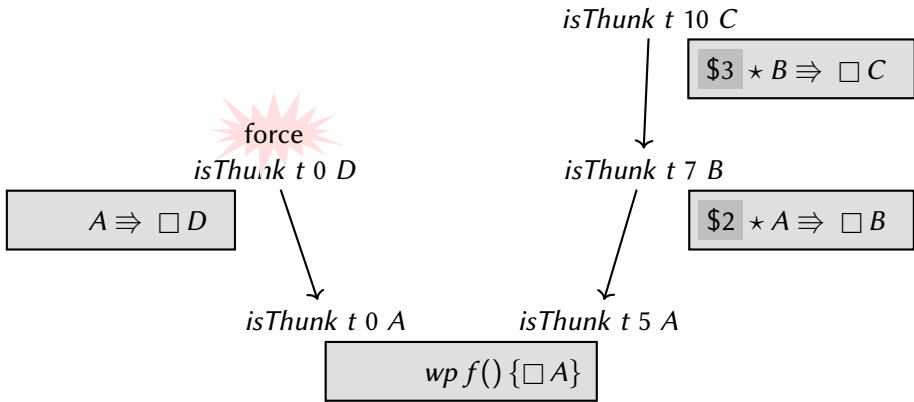
Example scenario:



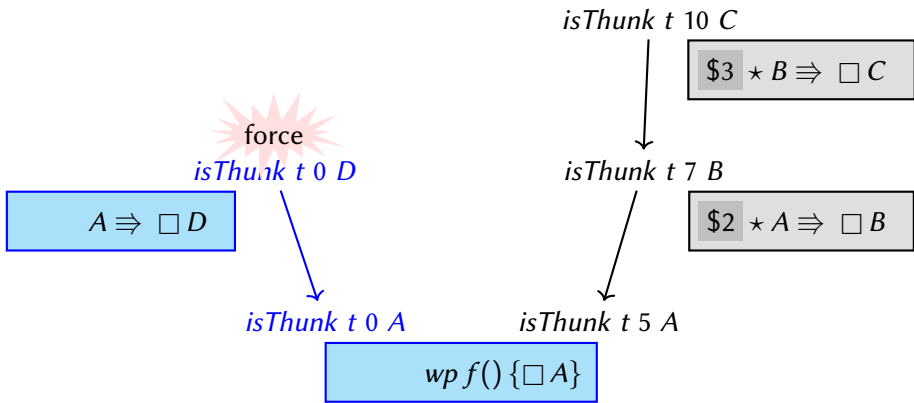
Example scenario:



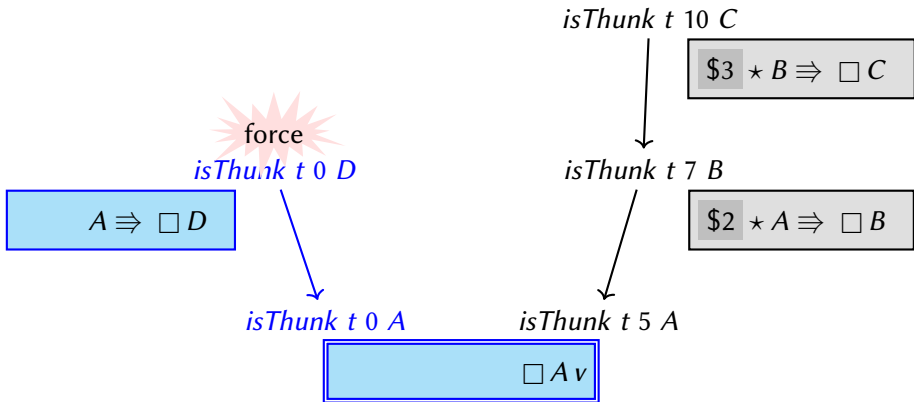
Example scenario:



Example scenario:

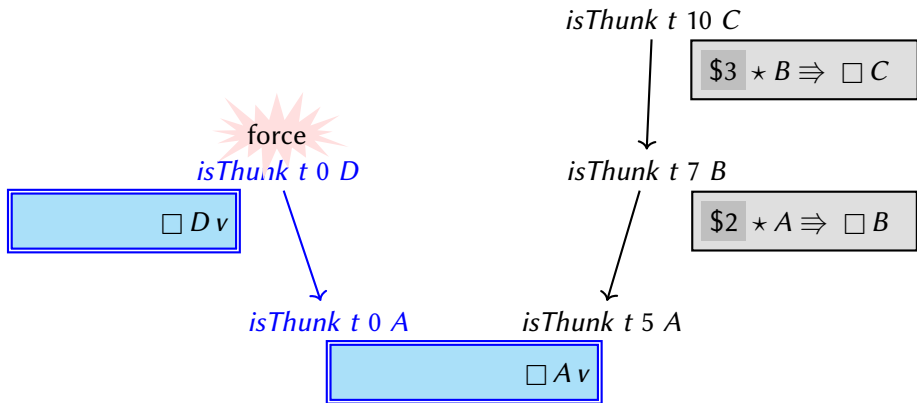


Example scenario:

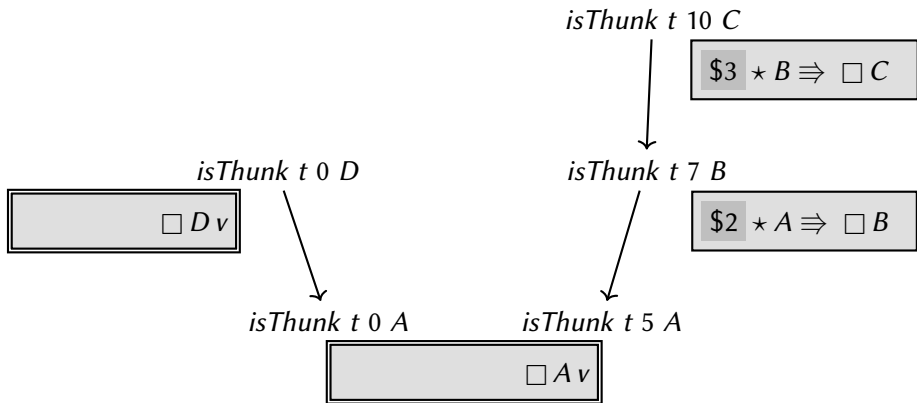




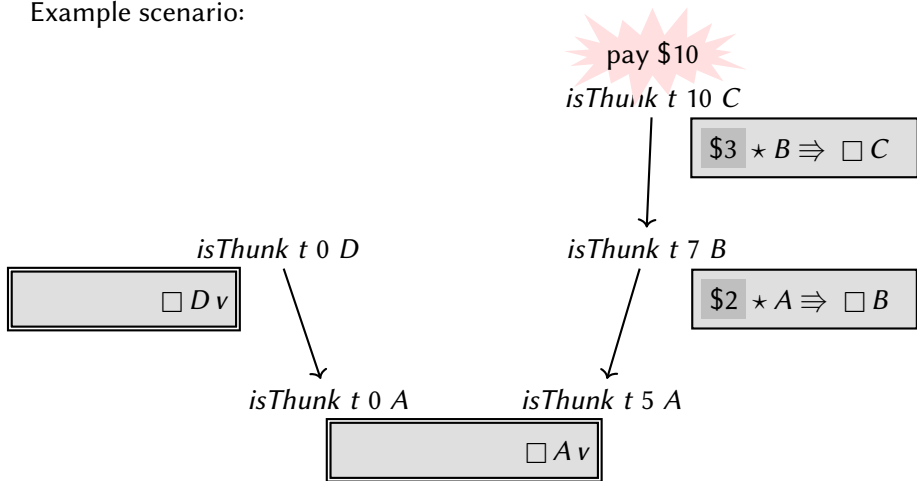
Example scenario:



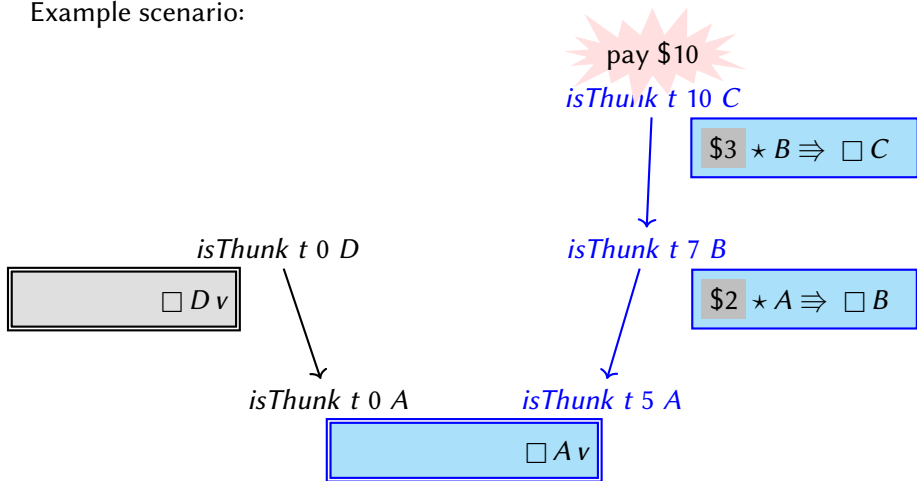
Example scenario:



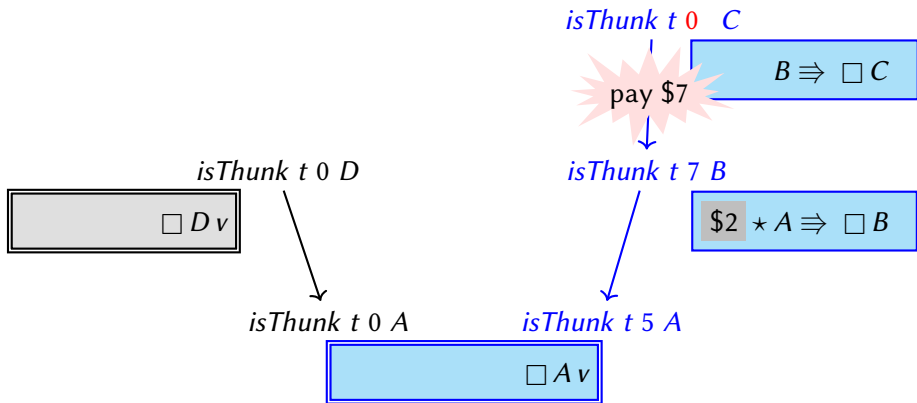
Example scenario:



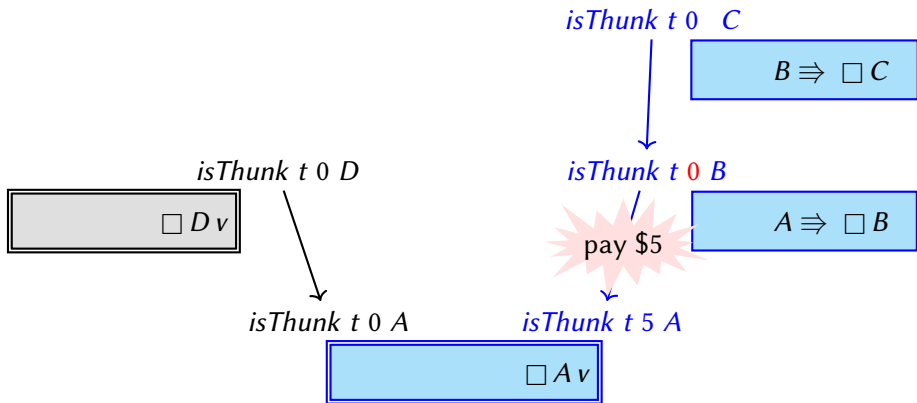
Example scenario:



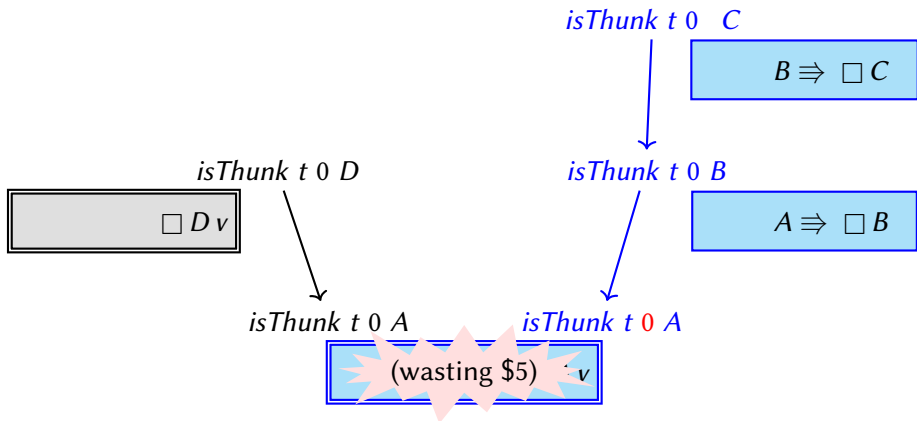
Example scenario:



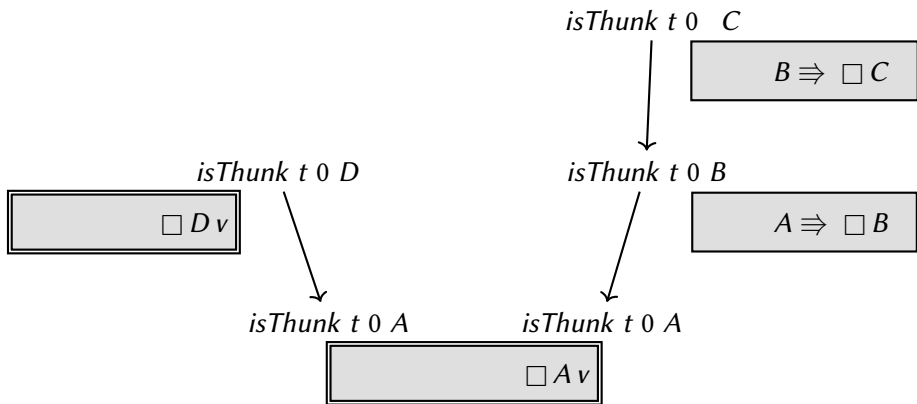
Example scenario:



Example scenario:

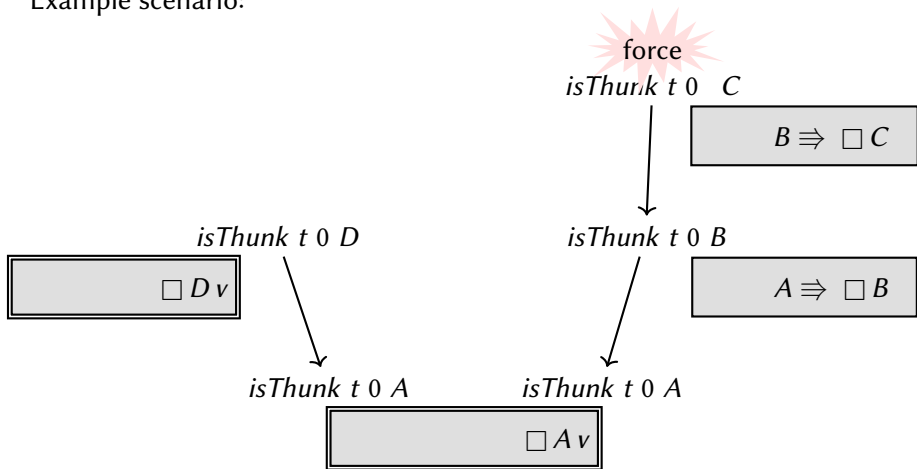


Example scenario:

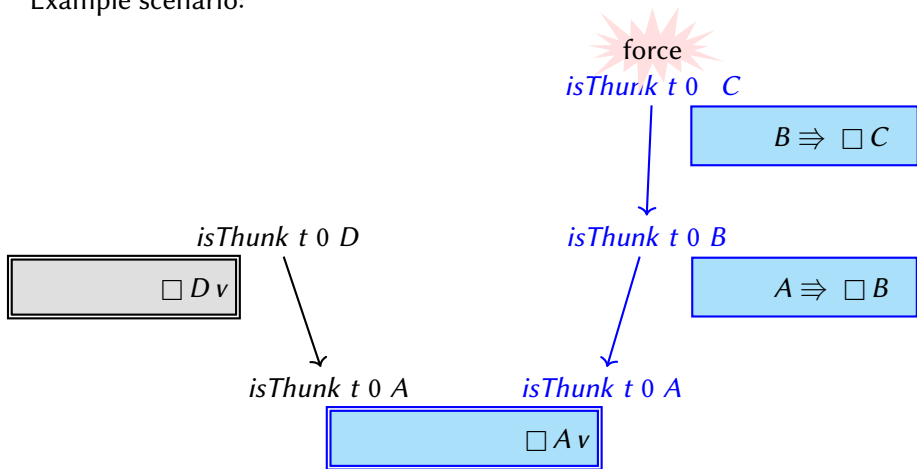




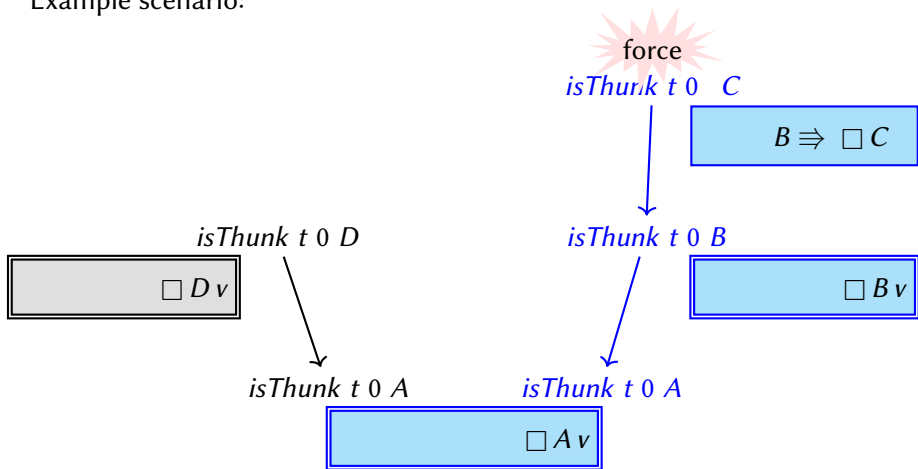
Example scenario:



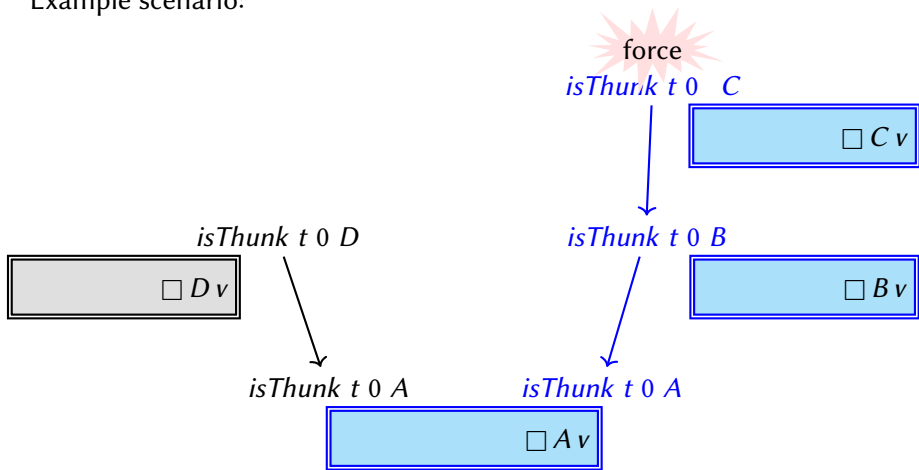
Example scenario:



Example scenario:



Example scenario:



We stack a new invariant and ghost state each time ANTICIPATE is used

Each height  $h \in \mathbb{N}$  has its own debit  $\gamma_{t,h}$

$$thinkInv\ t\ \varphi \triangleq \exists n. \boxed{\bullet n}^{\gamma_{t,0}} \star \bigvee \begin{cases} \exists f. t \mapsto Future\ f \star (\$n \ast wp\ f() \{ \square \varphi \}) \\ t \mapsto Busy \\ \exists v. t \mapsto Done\ v \star \square \varphi\ v \end{cases}$$

$$csqInv_h\ t\ \varphi\ \psi \triangleq \exists n. \boxed{\bullet n}^{\gamma_{t,h}} \star \bigvee \begin{cases} \forall v. \$n \star \varphi\ v \Rightarrow \square \psi\ v \\ \square \psi\ v \end{cases}$$

$$isThink_0\ t\ m\ \varphi \triangleq \boxed{\circ m}^{\gamma_{t,0}} \star \boxed{thinkInv\ t\ \varphi}$$

$$isThink_h\ t\ m\ \varphi \triangleq \exists m', \psi. m' \leq m \star \boxed{\circ m'}^{\gamma_{t,h}} \star \boxed{csqInv_h\ t\ \psi\ \varphi} \\ \star isThink_{h-1}\ t\ (m-m')\ \psi$$

$$isThink\ t\ m\ \varphi \triangleq \exists h. isThink_h\ t\ m\ \varphi$$

We stack a new invariant and ghost state each time ANTICIPATE is used

Each height  $h \in \mathbb{N}$  has its own debit  $\gamma_{t,h}$

$$\mathit{thinkInv} \ t \ \varphi \triangleq \exists n. \boxed{\bullet n}^{\gamma_{t,0}} \star \bigvee \begin{cases} \exists f. t \mapsto \mathit{Future} \ f \star (\$n \ast \mathit{wp} \ f() \ \{\square \varphi\}) \\ t \mapsto \mathit{Busy} \\ \exists v. t \mapsto \mathit{Done} \ v \star \square \varphi \ v \end{cases}$$

$$\mathit{csqInv}_h \ t \ \varphi \ \psi \triangleq \exists n. \boxed{\bullet n}^{\gamma_{t,h}} \star \bigvee \begin{cases} \forall v. \$n \star \varphi \ v \Rightarrow \square \psi \ v \\ \square \psi \ v \end{cases}$$

$$\mathit{isThink}_0 \ t \ m \ \varphi \triangleq \boxed{\circ m}^{\gamma_{t,0}} \star \boxed{\mathit{thinkInv} \ t \ \varphi}$$

$$\mathit{isThink}_h \ t \ m \ \varphi \triangleq \exists m', \psi. m' \leq m \star \boxed{\circ m'}^{\gamma_{t,h}} \star \boxed{\mathit{csqInv}_h \ t \ \psi \ \varphi} \\ \star \mathit{isThink}_{h-1} \ t \ (m - m') \ \psi$$

$$\mathit{isThink} \ t \ m \ \varphi \triangleq \exists h. \mathit{isThink}_h \ t \ m \ \varphi$$

**Omitted:** ghost state in  $\text{AUTH}(\text{EX}()) + \text{AG}(\text{VAL}())$  for remembering the value computed

# Actual implementation of thunks

```
type 'α thunk = 'α thunk_contents ref
and 'α thunk_contents =
  | Future of (unit → 'α)
  | Done of 'α

let create f =
  ref (Future f)

let force t =
  match ! t with
  | Future f →
    let v = f () in   - evaluate the thunk
    t := Done v ;     - memoize the result
    v
  | Done v →
    v
```

No reentrancy detection (2 states only)  $\implies$  static proof obligations

$$\begin{array}{cc}
 \{\$K_{cr} \star (\$n \multimap wp f() \{ \square \varphi \})\} & \{\$K_{frc} \star isThink t 0 \varphi\} \\
 \text{create } f & \text{force } t \\
 \{\lambda t. isThink t n \varphi\} & \{\lambda v. \varphi v\}
 \end{array}$$

PERSIST

$$\text{persistent}(isThink t m \varphi)$$

OVERESTIMATE

$$\frac{isThink t m_1 \varphi \quad m_1 \leq m_2}{isThink t m_2 \varphi}$$

PAY

$$\frac{isThink t m \varphi \quad \$p}{\Rightarrow isThink t (m - p) \varphi}$$

ANTICIPATE

$$\frac{isThink t m \varphi \quad \forall v. \$n \star \varphi v \Rightarrow \square \psi v}{\Rightarrow isThink t (m + n) \psi}$$



$$\{ \$K_{cr} \star (\$n \rightarrow * wp f() \{ \square \varphi \}) \} \quad \{ \$K_{frc} \star isThink t 0 \varphi \}$$

*create f*
*force t*

$$\{ \lambda t. isThink t n \varphi \} \quad \{ \lambda v. \varphi v \}$$

PERSIST

A thunk is evaluated only once:  
these arrows need not be persistent

OVERESTIMATE

$$\frac{isThink t m_1 \varphi \quad m_1 \leq m_2}{isThink t m_2 \varphi}$$

$$\frac{isThink t m \varphi \quad \$p}{\Rightarrow isThink t (m - p) \varphi}$$

ANTICIPATE

$$\frac{isThink t m \varphi \quad \forall v. \$n \star \varphi v \Rightarrow \square \psi v}{\Rightarrow isThink t (m + n) \psi}$$

$$\begin{array}{cc}
 \{\$K_{cr} \star (\$n \rightarrow * wp f() \{ \square \varphi \})\} & \{\$K_{frc} \star isThink t 0 \varphi\} \\
 \text{create } f & \text{force } t \\
 \{\lambda t. isThink t n \varphi\} & \{\lambda v. \varphi v\}
 \end{array}$$

PERSIST

$\text{persistent}(isThink t m \varphi)$

OVERESTIMATE

$$\frac{isThink t m_1 \varphi \quad m_1 \leq m_2}{isThink t m_2 \varphi}$$

PAY

$$\frac{isThink t m \varphi \quad \$p}{\Rightarrow isThink t (m - p) \varphi}$$

ANTICIPATE

$$\frac{isThink t m \varphi \quad \forall v. \$n \star \varphi v \Rightarrow \square \psi v}{\Rightarrow isThink t (m + n) \psi}$$

$$\begin{array}{cc}
 \{\$K_{cr} \star (\$n \multimap wp f() \{\square \varphi\})\} & \{\$K_{frc} \star isThink t 0 \varphi\} \\
 \text{create } f & \text{force } t \\
 \{\lambda t. isThink t n \varphi\} & \{\lambda v. \varphi v\}
 \end{array}$$

PERSIST

$persist(isThink t m \varphi)$

Reentrancy?

OVERESTIMATE

$$\frac{isThink t m_1 \varphi \quad m_1 \leq m_2}{isThink t m_2 \varphi}$$

PAY

$$\frac{isThink t m \varphi \quad \$p}{\Rightarrow isThink t (m - p) \varphi}$$

ANTICIPATE

$$\frac{isThink t m \varphi \quad \forall v. \$n \star \varphi v \Rightarrow \square \psi v}{\Rightarrow isThink t (m + n) \psi}$$

One *canForce* token exists  
at the beginning of the world

$$\frac{\text{CANFORCEEXCL} \quad \text{canForce} \quad \text{canForce}}{\text{False}}$$

$$\{\$K_{\text{cr}} \star (\$n \rightarrow * \text{wp } f() \{ \square \varphi \})\}$$

*create f*

$$\{\lambda t. \text{isThunk } t \ n \ \varphi\}$$

$$\{\$K_{\text{frc}} \star \text{isThunk } t \ 0 \ \varphi \star \text{canForce}\}$$

*force t*

$$\{\lambda v. \varphi \ v \ \star \text{canForce}\}$$

PERSIST

*persistent(isThunk t m φ)*

OVERESTIMATE

$$\frac{\text{isThunk } t \ m_1 \ \varphi \quad m_1 \leq m_2}{\text{isThunk } t \ m_2 \ \varphi}$$

PAY

$$\frac{\text{isThunk } t \ m \ \varphi \quad \$p}{\Rightarrow \text{isThunk } t \ (m - p) \ \varphi}$$

ANTICIPATE

$$\frac{\text{isThunk } t \ m \ \varphi \quad \forall v. \$n \star \varphi \ v \Rightarrow \square \psi \ v}{\Rightarrow \text{isThunk } t \ (m + n) \ \psi}$$

One *canForce* token exists  
at the beginning of the world

$$\frac{\text{CANFORCEEXCL} \quad \text{canForce} \quad \text{canForce}}{\text{False}}$$

$$\frac{\{\$K_{cr} \star (\$n \rightarrow * wp f() \{ \square \varphi \})\} \quad \{\$K_{frc} \star isThink t 0 \varphi \star canForce\}}{\text{create } f \quad \text{force } t}$$

$$\frac{\{\lambda t. isThink t n \varphi\} \quad \{\lambda v. \varphi v \star canForce\}}{\text{persistent}(isThink t m \varphi)}$$

How to force a think from another think?

$$\frac{\text{OVERESTIMATE} \quad isThink t m_1 \varphi \quad m_1 \leq m_2}{isThink t m_2 \varphi}$$

$$\frac{\text{PAY} \quad isThink t m \varphi \quad \$p}{\Rightarrow isThink t (m - p) \varphi}$$

$$\frac{\text{ANTICIPATE} \quad isThink t m \varphi \quad \forall v. \$n \star \varphi v \Rightarrow \square \psi v}{\Rightarrow isThink t (m + n) \psi}$$

One *canForce*  $\top$  token exists  
at the beginning of the world

$$\frac{\text{CANFORCEEXCL} \quad \text{canForce } \mathcal{N}_1 \quad \text{canForce } \mathcal{N}_2}{(\uparrow \mathcal{N}_1) \cap (\uparrow \mathcal{N}_2) = \emptyset}$$

$$\{\$K_{\text{cr}} \star (\$n \star \text{wp } f() \{ \square \varphi \})\}$$

*create f*

$$\{\lambda t. \text{isThink } t \mathcal{N} n \varphi\}$$

$$\{\$K_{\text{frc}} \star \text{isThink } t \mathcal{N} 0 \varphi \star \text{canForce } \mathcal{N}\}$$

*force t*

$$\{\lambda v. \varphi v \star \text{canForce } \mathcal{N}\}$$

PERSIST

$$\text{persistent}(\text{isThink } t \mathcal{N} m \varphi)$$

OVERESTIMATE

$$\frac{\text{isThink } t \mathcal{N} m_1 \varphi \quad m_1 \leq m_2}{\text{isThink } t \mathcal{N} m_2 \varphi}$$

PAY

$$\frac{\text{isThink } t \mathcal{N} m \varphi \quad \$p}{\Rightarrow \text{isThink } t \mathcal{N} (m - p) \varphi}$$

ANTICIPATE

$$\frac{\text{isThink } t \mathcal{N} m \varphi \quad \forall v. \$n \star \varphi v \Rightarrow \square \psi v}{\Rightarrow \text{isThink } t \mathcal{N} (m + n) \psi}$$

One *canForce*  $\top$  token exists  
at the beginning of the world

$$\frac{\text{CANFORCEEXCL} \quad \text{canForce } \mathcal{N}_1 \quad \text{canForce } \mathcal{N}_2}{(\uparrow \mathcal{N}_1) \cap (\uparrow \mathcal{N}_2) = \emptyset}$$

$$\left\{ \$K_{\text{cr}} \star (\$n \star \text{wp } f() \{ \square \varphi \}) \right\} \quad \left\{ \$K_{\text{frc}} \star \text{isThink } t \mathcal{N} 0 \varphi \star \text{canForce } \mathcal{N} \right\}$$

$$\text{create } f \quad \text{force } t$$

$$\left\{ \lambda t. \text{isThink } t \mathcal{N} n \varphi \right\} \quad \left\{ \lambda v. \varphi v \star \text{canForce } \mathcal{N} \right\}$$

...But how to thread the token to the inner thunk?

$$\frac{\text{PERSIST}}{\text{persistent}(\text{isThink } t \mathcal{N} m \varphi)}$$

$$\frac{\text{OVERESTIMATE} \quad \text{isThink } t \mathcal{N} m_1 \varphi \quad m_1 \leq m_2}{\text{isThink } t \mathcal{N} m_2 \varphi}$$

$$\frac{\text{PAY} \quad \text{isThink } t \mathcal{N} m \varphi \quad \$p}{\Rightarrow \text{isThink } t \mathcal{N} (m - p) \varphi}$$

$$\frac{\text{ANTICIPATE} \quad \text{isThink } t \mathcal{N} m \varphi \quad \forall v. \$n \star \varphi v \Rightarrow \square \psi v}{\Rightarrow \text{isThink } t \mathcal{N} (m + n) \psi}$$

One *canForce*  $\top$  token exists  
at the beginning of the world

$$\frac{\text{CANFORCEEXCL} \quad \text{canForce } \mathcal{N}_1 \quad \text{canForce } \mathcal{N}_2}{(\uparrow \mathcal{N}_1) \cap (\uparrow \mathcal{N}_2) = \emptyset}$$

$$\begin{array}{c} \$K_{\text{cr}} \star (\$n \star R \text{ create } f \text{ } \{ \square \varphi \star R \}) \{ \$K_{\text{frc}} \star \text{isThink } t \mathcal{N} \ 0 \ R \ \varphi \star \text{canForce } \mathcal{N} \star R \\ \{ \lambda t. \text{isThink } t \ \mathcal{N} \ n \ R \ \varphi \} \quad \quad \quad \{ \lambda v. \varphi \ v \star \text{canForce } \mathcal{N} \star R \} \end{array}$$

PERSIST

$$\text{persistent}(\text{isThink } t \ \mathcal{N} \ m \ R \ \varphi)$$

OVERESTIMATE

$$\frac{\text{isThink } t \ \mathcal{N} \ m_1 \ R \ \varphi \quad m_1 \leq m_2}{\text{isThink } t \ \mathcal{N} \ m_2 \ R \ \varphi}$$

PAY

$$\frac{\text{isThink } t \ \mathcal{N} \ m \ R \ \varphi \quad \$p}{\Rightarrow \text{isThink } t \ \mathcal{N} \ (m - p) \ R \ \varphi}$$

ANTICIPATE

$$\frac{\text{isThink } t \ \mathcal{N} \ m \ R \ \varphi \quad \forall v. \$n \star \varphi \ v \star R \Rightarrow \square \psi \ v \star R}{\Rightarrow \text{isThink } t \ \mathcal{N} \ (m + n) \ R \ \psi}$$



# Implementation of streams

**type** 'α stream = 'α cell *thunk* – a stream is computed on-demand  
**and** 'α cell = Nil | Cons **of** 'α × 'α stream

**let** pop (xs : 'α stream) =  
  **match** *Thunk.force* xs **with**  
  | Cons (x, xs') → Some (x, xs')  
  | Nil → None

**let rec** append (xs : 'α stream) (ys : 'α stream) =  
  *Thunk.create@@fun*()→ – this thunk has a constant overhead  
  **match** *Thunk.force* xs **with**  
  | Cons (x, xs') → Cons (x, append xs' ys)  
  | Nil → *Thunk.force* ys

**let** rev\_of\_list (xs : 'α list) : 'α stream =  
  **let rec** rev\_app (xs : 'α list) (ys : 'α cell) = – rev\_app reverses the list eagerly  
    **match** xs **with** – ↓ these new thunks have cost 0  
    | x :: xs' → rev\_app xs' (Cons (x, *Thunk.create@@fun*()→ ys))  
    | [] → ys **in**  
  *Thunk.create@@fun*()→ *rev\_app* xs Nil – this leading thunk is costly

## (Selected rules) Specification of streams

$$\{\$K_{\text{ap}} \star \text{isStream } s [m_1, \dots, m_n] [v_1, \dots, v_n] \star \text{isStream } s' [m'_1, \dots, m'_n] [v'_1, \dots, v'_n]\}$$
$$\text{append } s \ s'$$
$$\{\lambda t. \text{isStream } t [A + m_1, \dots, A + m_n, m'_1, \dots, m'_n] [v_1, \dots, v_n, v'_1, \dots, v'_n]\}$$

$$\{\$K_{\text{rv}} \star \text{isList } \ell [v_1, \dots, v_n]\}$$
$$\text{rev\_of\_list } \ell$$
$$\{\lambda s. \text{isStream } s [B \cdot n, 0, \dots, 0] [v_n, \dots, v_1]\}$$

PAYSTREAM

$$\frac{\text{isStream } s [m_1, m_2, \dots, m_n] [v_1, \dots, v_n] \quad \$p}{\Rightarrow \text{isStream } s [m_1 - p, m_2, \dots, m_n] [v_1, \dots, v_n]}$$

ANTICIPATE+OVERESTIMATESTREAM

$$\frac{\text{isStream } s [m_1, \dots, m_n] [v_1, \dots, v_n] \quad \forall k. \sum_{i \leq k} m_i \leq \sum_{i \leq k} m'_i}{\Rightarrow \text{isStream } s [m'_1, \dots, m'_n] [v_1, \dots, v_n]}$$

We forbid recursive streams by using **generations**  $g \in \mathbb{N}$ :

$$\text{isStream } s [m_1, \dots, m_n] [v_1, \dots, v_n] \triangleq$$

$$\exists g_1. \text{isThunk } s \mathcal{N}_{g_1} m_1 (\text{naInvTok } \mathcal{E}_{g_1}) (\lambda c_1. \exists s_2. c_1 = \text{Cons}(v_1, s_2) \star$$

$$\exists g_2 \leq g_1. \text{isThunk } s_2 \mathcal{N}_{g_2} m_2 (\text{naInvTok } \mathcal{E}_{g_2}) (\lambda c_2. \exists s_3. c_2 = \text{Cons}(v_2, s_3) \star$$

$\dots$

$$\exists g_{n+1} \leq g_n. \text{isThunk } s_{n+1} \mathcal{N}_{g_{n+1}} 0 (\text{naInvTok } \mathcal{E}_{g_{n+1}}) (\lambda c_{n+1}. c_{n+1} =$$

where:

$$\mathcal{E}_g \triangleq \top \setminus \uparrow \mathcal{N}_g$$

$$\mathcal{E}_g \subseteq \mathcal{E}_{g+1}$$

$$\uparrow \mathcal{N}_{g+1} \subseteq \uparrow \mathcal{N}_g$$