Space-Efficient Fragments of Higher-Order Fixpoint Logic

Florian Bruse [1 2] Martin Lange [1] Etienne Lozes [2]

[1]Universität Kassel, Germany

[2]LSV, ENS Paris-Saclay, France

September 8, 2017

Motivation

Higher-order Modal Fixpoint Logic:

- extension of the modal μ-calculus by simply typed λ-calculus
- very expressive logic, e.g., context-sensitive path languages, assume-guarantee properties
- captures time hierarchy: type order $k+1$ model checking is $k+1$-EXPTIME complete (via alternating reachability game)

Motivation

Higher-order Modal Fixpoint Logic:

- extension of the modal μ-calculus by simply typed λ-calculus
- very expressive logic, e.g., context-sensitive path languages, assume-guarantee properties
- captures time hierarchy: type order $k+1$ model checking is $k+1$-EXPTIME complete (via alternating reachability game)

Question: Can we capture space hierarchy?

Motivation

Higher-order Modal Fixpoint Logic:

- extension of the modal μ-calculus by simply typed λ-calculus
- very expressive logic, e.g., context-sensitive path languages, assume-guarantee properties
- captures time hierarchy: type order $k+1$ model checking is $k+1$-EXPTIME complete (via alternating reachability game)

Question: Can we capture space hierarchy?

$\rightarrow$ Tail-recursive fragment:

- restrict interplay of fixpoints, mixing with certain operators
- model-checking for order $k+1$ in k-EXPSPACE (via nondet. reachability game)

Motivation

Higher-order Modal Fixpoint Logic:

- extension of the modal μ-calculus by simply typed λ-calculus
- very expressive logic, e.g., context-sensitive path languages, assume-guarantee properties
- captures time hierarchy: type order $k+1$ model checking is $k+1$-EXPTIME complete (via alternating reachability game)

Question: Can we capture space hierarchy?

$\rightarrow$ Tail-recursive fragment:

- restrict interplay of fixpoints, mixing with certain operators
- model-checking for order $k+1$ in k-EXPSPACE (via nondet. reachability game)
- matching lower bound via reduction to order-k corridor tiling problem

Motivation

Higher-order Modal Fixpoint Logic:

- extension of the modal μ-calculus by simply typed λ-calculus
- very expressive logic, e.g., context-sensitive path languages, assume-guarantee properties
- captures time hierarchy: type order $k+1$ model checking is $k+1$-EXPTIME complete (via alternating reachability game)

Question: Can we capture space hierarchy?

$\rightarrow$ Tail-recursive fragment:

- restrict interplay of fixpoints, mixing with certain operators
- model-checking for order $k+1$ in k-EXPSPACE (via nondet. reachability game)
- matching lower bound via reduction to order-k corridor tiling problem
- still very expressive: $a^n b^n c^n$ path language

Syntax of HFL

HFL $=$ modal μ-calculus

$$\varphi \quad ::= \quad P \mid X \qquad \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle a \rangle \varphi \mid \mu\, X \qquad .\varphi$$

plus duals $\varphi \wedge \psi$, $[a]\varphi$, $\nu\, X$ *natively*

Syntax of HFL

HFL = modal μ-calculus + simply typed λ-calculus

[Viswanathan^2 '04]

$$\varphi \quad ::= \quad P \mid X \mid x \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle a \rangle \varphi \mid \mu\, X \,.\varphi \mid \lambda\, x \,.\varphi \mid \varphi\ \varphi$$

plus duals $\varphi \wedge \psi$, $[a]\varphi$, $\nu\, X$ *natively*

Syntax of HFL

HFL = modal μ-calculus + simply typed λ-calculus

[Viswanathan^2 '04]

$$\varphi \quad ::= \quad P \mid X \mid x \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle a \rangle \varphi \mid \mu\, X \,.\varphi \mid \lambda\, x \,.\varphi \mid \varphi\ \varphi$$

plus duals $\varphi \wedge \psi$, $[a]\varphi$, $\nu\ X$ *natively*

But expressions like $\langle a \rangle P \langle b \rangle P$ not well defined

Syntax of HFL

HFL = modal μ-calculus + simply typed λ-calculus

[Viswanathan^2 '04]

$$\varphi \quad ::= \quad P \mid X \mid x \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle a \rangle \varphi \mid \mu(X : \tau).\varphi \mid \lambda(x : \tau).\varphi \mid \varphi\ \varphi$$

plus duals $\varphi \wedge \psi$, $[a]\varphi$, $\nu(X : \tau)$ *natively*

But expressions like $\langle a \rangle P \langle b \rangle P$ not well defined

well-formedness condition given by type system

Types

simple types given via $\tau ::= \mathsf{Pr}$

complete lattice over transition system

$\mathcal{T} = (\mathcal{S}, \to, L)$

$[\![\mathsf{Pr}]\!] := (2^{\mathcal{S}}, \subseteq)$

Types

simple types given via $\tau ::= \mathsf{Pr} \mid \tau \to \tau$

because of right-associativity: $\tau = \tau_1 \to \ldots \to \tau_m \to \mathsf{Pr}$

each type induces a complete lattice over transition system $\mathcal{T} = (\mathcal{S}, \to, L)$ using pointwise orderings $\sqsubseteq$

$$\begin{aligned} [\![\mathsf{Pr}]\!] &:= (2^{\mathcal{S}}, \subseteq) \\ [\![\sigma \to \tau]\!] &:= ([\![\sigma]\!] \to [\![\tau]\!], \sqsubseteq) \end{aligned}$$

Semantics by Example

Consider $(\mu X.\lambda x.\, x \vee (X\,[a]x))\,P$.

Semantics by Example

Consider $(\mu X.\lambda x.\, x \vee (X\,[a]x))\,P$.

Unfolding via $\sigma X.\psi = \psi[\sigma X.\psi/X]$ yields
$\Big(\lambda x.\, x \vee \big(\mu X.\lambda x'.\, x' \vee (X\,[a]x')\big)\,[a]x\Big)\,P$.

Semantics by Example

Consider $(\mu X.\lambda x.\, x \vee (X\,[a]x))\,P$.

Unfolding via $\sigma X.\psi = \psi[\sigma X.\psi/X]$ yields
$\left(\lambda x.\, x \vee \left(\mu X.\lambda x'.\, x' \vee (X\,[a]x')\right)[a]x\right) P$.

Using β-reduction we get
$P \vee \left(\mu X.\lambda x'.\, x' \vee (X\,[a]x')\right)[a]P$.

Semantics by Example

Consider $(\mu X.\lambda x.\, x \vee (X\,[a]x))\,P$.

Unfolding via $\sigma X.\psi = \psi[\sigma X.\psi/X]$ yields
$\Big(\lambda x.\, x \vee \big(\mu X.\lambda x'.\, x' \vee (X\,[a]x')\big)\,[a]x\Big)\,P$.

Using β-reduction we get
$P \vee \big(\mu X.\lambda x'.\, x' \vee (X\,[a]x')\big)\,[a]P$.

More unfolding:
$P \vee (\lambda x'.\, x' \vee \big(\mu X.\lambda x''.\, x'' \vee (X\,[a]x'')\big)\,[a]x')\,[a]P$.

Semantics by Example

Consider $(\mu X.\lambda x.\, x \vee (X\,[a]x))\,P$.

Unfolding via $\sigma X.\psi = \psi[\sigma X.\psi/X]$ yields
$\left(\lambda x.\, x \vee \left(\mu X.\lambda x'.\, x' \vee (X\,[a]x')\right)[a]x\right) P$.

Using β-reduction we get
$P \vee \left(\mu X.\lambda x'.\, x' \vee (X\,[a]x')\right)[a]P$.

More unfolding:
$P \vee (\lambda x'.\, x' \vee \left(\mu X.\lambda x''.\, x'' \vee (X\,[a]x'')\right)[a]x')\,[a]P$.

More β-reduction:
$P \vee [a]P \vee \left(\mu X.\lambda x''.\, x'' \vee (X\,[a]x'')\right)[a][a]P)$.

Semantics by Example

Consider $(\mu X.\lambda x.\, x \vee (X\,[a]x))\,P$.

Unfolding via $\sigma X.\psi = \psi[\sigma X.\psi/X]$ yields
$\left(\lambda x.\, x \vee \left(\mu X.\lambda x'.\, x' \vee (X\,[a]x')\right)[a]x\right) P$.

Using β-reduction we get
$P \vee \left(\mu X.\lambda x'.\, x' \vee (X\,[a]x')\right)[a]P$.

More unfolding:
$P \vee (\lambda x'.\, x' \vee \left(\mu X.\lambda x''.\, x'' \vee (X\,[a]x'')\right)[a]x')\,[a]P$.

More β-reduction:
$P \vee [a]P \vee \left(\mu X.\lambda x''.\, x'' \vee (X\,[a]x'')\right)[a][a]P)$.

We get: $P \vee [a]P \vee [a][a]P \vee \ldots = \bigvee_{i=0}^{n} [a]^i P$

uniform inevitability!

Tail-Recursion

Limit use of fp vars: No free fixpoint variables in all subformulas of certain form

- Form $\varphi \wedge \psi$: no free fixpoint variables in φ
- Form $[a]\varphi$: no free fixpoint variables in φ
- Form $\neg\varphi$: no free fixpoint variables in φ
- Form $\psi\,\varphi$: no free fixpoint variables in φ

Tail-Recursion

Limit use of fp vars: No free fixpoint variables in all subformulas of certain form

- Form $\varphi \wedge \psi$: no free fixpoint variables in φ
- Form $[a]\varphi$: no free fixpoint variables in φ
- Form $\neg\varphi$: no free fixpoint variables in φ
- Form $\psi\,\varphi$: no free fixpoint variables in φ

Result: free fixpoint variables not under any of these operators, each fixpoint definition "morally" contains only one variable (modulo nondeterminism), in operator position

Tail-Recursion

Limit use of fp vars: No free fixpoint variables in all subformulas of certain form

- Form $\varphi \wedge \psi$: no free fixpoint variables in φ
- Form $[a]\varphi$: no free fixpoint variables in φ
- Form $\neg\varphi$: no free fixpoint variables in φ
- Form $\psi\,\varphi$: no free fixpoint variables in φ

Result: free fixpoint variables not under any of these operators, each fixpoint definition "morally" contains only one variable (modulo nondeterminism), in operator position

$\mu X.p \vee (\langle a \rangle (X \wedge q))$ not tail-recursive

Tail-Recursion

Limit use of fp vars: No free fixpoint variables in all subformulas of certain form

- Form $\varphi \wedge \psi$: no free fixpoint variables in φ
- Form $[a]\varphi$: no free fixpoint variables in φ
- Form $\neg\varphi$: no free fixpoint variables in φ
- Form $\psi\,\varphi$: no free fixpoint variables in φ

Result: free fixpoint variables not under any of these operators, each fixpoint definition "morally" contains only one variable (modulo nondeterminism), in operator position

$\mu X.p \vee (\langle a \rangle (X \wedge q))$ not tail-recursive

$\mu F.\lambda x.[\cdot]\bot \vee (F\,[\cdot]x))\bot$ tail recursive

Tail-Recursion

Limit use of fp vars: No free fixpoint variables in all subformulas of certain form

- Form $\varphi \wedge \psi$: no free fixpoint variables in φ
- Form $[a]\varphi$: no free fixpoint variables in φ
- Form $\neg\varphi$: no free fixpoint variables in φ
- Form $\psi\,\varphi$: no free fixpoint variables in φ

Result: free fixpoint variables not under any of these operators, each fixpoint definition "morally" contains only one variable (modulo nondeterminism), in operator position

$\mu X.p \vee (\langle a \rangle (X \wedge q))$ not tail-recursive

$\mu F.\lambda x.[\cdot]\bot \vee (F\,[\cdot]x))\bot$ tail recursive

Thm.: The order $(k+1)$ model-checking problem for tail-recursive HFL is k-EXPSPACE-complete.

The Upper Bound - Preparations

Want algorithm in k-fold exponential space $\rightarrow$ get nondeterminism for free (Savitch's Theorem)

The Upper Bound - Preparations

Want algorithm in k-fold exponential space $\rightarrow$ get nondeterminism for free (Savitch's Theorem)

Consider $\mu(X : \tau).\varphi$:

Define $X^0 = \bot_\tau; X^{i+1} = \varphi[X^i/X]$

The Upper Bound - Preparations

Want algorithm in k-fold exponential space $\rightarrow$ get nondeterminism for free (Savitch's Theorem)

Consider $\mu(X : \tau).\varphi$:

Define $X^0 = \bot_\tau; X^{i+1} = \varphi[X^i/X]$

Over given LTS, $\mu(: \tau)X.\varphi \equiv X^m$ for some $m = \mathrm{ht}(\tau)$ that is $k+1$-fold exponential in size of LTS

The Upper Bound - Preparations

Want algorithm in k-fold exponential space $\rightarrow$ get nondeterminism for free (Savitch's Theorem)

Consider $\mu(X : \tau).\varphi$:

Define $X^0 = \bot_\tau; X^{i+1} = \varphi[X^i/X]$

Over given LTS, $\mu(: \tau)X.\varphi \equiv X^m$ for some $m = \mathrm{ht}(\tau)$ that is $k+1$-fold exponential in size of LTS
$\rightarrow$ can be stored with k-fold exponentially many bits

same for greatest fixpoints

The Upper Bound - Preparations

Want algorithm in k-fold exponential space $\rightarrow$ get nondeterminism for free (Savitch's Theorem)

Consider $\mu(X : \tau).\varphi$:

Define $X^0 = \bot_\tau; X^{i+1} = \varphi[X^i/X]$

Over given LTS, $\mu(: \tau)X.\varphi \equiv X^m$ for some $m = \mathrm{ht}(\tau)$ that is $k+1$-fold exponential in size of LTS
$\rightarrow$ can be stored with k-fold exponentially many bits

same for greatest fixpoints

Approach: use top-down local model-checking

- cover fixpoints by unfolding

The Upper Bound - Preparations

Want algorithm in k-fold exponential space $\rightarrow$ get nondeterminism for free (Savitch's Theorem)

Consider $\mu(X : \tau).\, \varphi$:

Define $X^0 = \bot_\tau; X^{i+1} = \varphi[X^i/X]$

Over given LTS, $\mu(: \tau)X.\, \varphi \equiv X^m$ for some $m = \mathrm{ht}(\tau)$ that is $k+1$-fold exponential in size of LTS
$\rightarrow$ can be stored with k-fold exponentially many bits

same for greatest fixpoints

Approach: use top-down local model-checking

- cover fixpoints by unfolding
- cover $\vee, \langle a \rangle$ by free nondeterminism

The Upper Bound - Preparations

Want algorithm in k-fold exponential space $\rightarrow$ get nondeterminism for free (Savitch's Theorem)

Consider $\mu(X : \tau).\, \varphi$:

Define $X^0 = \bot_\tau; X^{i+1} = \varphi[X^i/X]$

Over given LTS, $\mu(: \tau)X.\, \varphi \equiv X^m$ for some $m = \mathrm{ht}(\tau)$ that is $k+1$-fold exponential in size of LTS
$\rightarrow$ can be stored with k-fold exponentially many bits

same for greatest fixpoints

Approach: use top-down local model-checking

- cover fixpoints by unfolding
- cover $\vee, \langle a \rangle$ by free nondeterminism
- cover $\wedge, [a], \neg$, application via tail-recursion

The Upper Bound

Model-checking whether state t in some LTS is model of ψ

procedure **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ consumes

- a state s

The Upper Bound

Model-checking whether state t in some LTS is model of ψ

procedure **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ consumes

- a state s
- a subformula φ of ψ

The Upper Bound

Model-checking whether state t in some LTS is model of ψ

procedure **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ consumes

- a state s
- a subformula φ of ψ
- a list $(f_1, \ldots, f_n)$ of arguments to φ

The Upper Bound

Model-checking whether state t in some LTS is model of ψ

procedure **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ consumes

- a state s
- a subformula φ of ψ
- a list $(f_1, \ldots, f_n)$ of arguments to φ
- an environment η for free variables in φ

The Upper Bound

Model-checking whether state t in some LTS is model of ψ

procedure **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ consumes

- a state s
- a subformula φ of ψ
- a list $(f_1, \ldots, f_n)$ of arguments to φ
- an environment η for free variables in φ
- a counter cnt specifying degree of unfolding

The Upper Bound

Model-checking whether state t in some LTS is model of ψ

procedure **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ consumes

- a state s
- a subformula φ of ψ
- a list $(f_1, \ldots, f_n)$ of arguments to φ
- an environment η for free variables in φ
- a counter cnt specifying degree of unfolding

Behavior of **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ depends on form of φ:

- If φ is atomic, return true if $s \models \varphi$, false else

The Upper Bound

Model-checking whether state t in some LTS is model of ψ

procedure **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ consumes

- a state s
- a subformula φ of ψ
- a list $(f_1, \ldots, f_n)$ of arguments to φ
- an environment η for free variables in φ
- a counter cnt specifying degree of unfolding

Behavior of **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ depends on form of φ:

- If φ is atomic, return true if $s \models \varphi$, false else
- If $\varphi = \langle a \rangle \varphi'$, guess t with $s \xrightarrow{a} t$, return **check**$(t, \varphi', (f_1, \ldots, f_n), \eta, \mathsf{cnt})$.

The Upper Bound

Model-checking whether state t in some LTS is model of ψ

procedure **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathrm{cnt})$ consumes

- a state s
- a subformula φ of ψ
- a list $(f_1, \ldots, f_n)$ of arguments to φ
- an environment η for free variables in φ
- a counter cnt specifying degree of unfolding

Behavior of **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathrm{cnt})$ depends on form of φ:

- If φ is atomic, return true if $s \models \varphi$, false else
- If $\varphi = \langle a \rangle \varphi'$, guess t with $s \xrightarrow{a} t$, return **check**$(t, \varphi', (f_1, \ldots, f_n), \eta, \mathrm{cnt})$.
- ...
- If $\varphi = \varphi_1 \wedge \varphi_2$, $\rightarrow \varphi_1$ has no free fp vars. Return false if **check**$(s, \varphi_1, (f_1, \ldots, f_n), \eta, \mathrm{cnt})$ returns false, return **check**$(s, \varphi_2, (f_1, \ldots, f_n), \eta, \mathrm{cnt})$ else

The Upper Bound (cont.)

Behavior of **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ depends on form of φ:

- If $\varphi = \neg\varphi'$, $\to \varphi'$ has no free fp vars. Return opposite of **check**$(s, \varphi', (f_1, \ldots, f_n), \eta, \mathsf{cnt})$

The Upper Bound (cont.)

Behavior of **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ depends on form of φ:

- If $\varphi = \neg\varphi'$, $\to \varphi'$ has no free fp vars. Return opposite of **check**$(s, \varphi', (f_1, \ldots, f_n), \eta, \mathsf{cnt})$
- If $\varphi = \varphi'\,\varphi''$, $\to \varphi''$ has no free fp vars. Compute $f_{n+1} =$ **buildFT**(φ'', η), return **check**$(s, \varphi', (f_1, \ldots, f_n, f_{n+1}), \eta, \mathsf{cnt})$

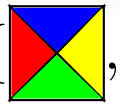
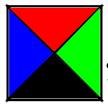
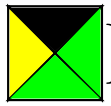
The Upper Bound (cont.)

Behavior of **check**$(s, \varphi, (f_1, \ldots, f_n), \eta, \mathsf{cnt})$ depends on form of φ:

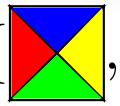
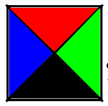
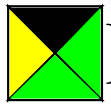
- If $\varphi = \neg\varphi'$, $\to \varphi'$ has no free fp vars. Return opposite of **check**$(s, \varphi', (f_1, \ldots, f_n), \eta, \mathsf{cnt})$
- If $\varphi = \varphi'\,\varphi''$, $\to \varphi''$ has no free fp vars. Compute $f_{n+1} =$ **buildFT**(φ'', η), return **check**$(s, \varphi', (f_1, \ldots, f_n, f_{n+1}), \eta, \mathsf{cnt})$
- $\ldots$
- If $\varphi = \mu(X : \tau).\,\varphi'$, return **check**$(s, \varphi', (f_1, \ldots, f_n), \eta, \mathsf{cnt}[X \mapsto \mathsf{ht}(\tau)])$.
- If $\varphi = X$, X least fp var, return false if $\mathsf{cnt}(X) = 0$, else return **check**$(s, \mathsf{fp}(X), (f_1, \ldots, f_n), \eta, \mathsf{cnt}[X\text{- - }]$.

buildFT(φ, η) calls **check**$(\ldots)$ again, iterating over all suitable arguments

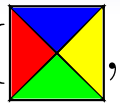
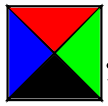
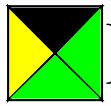
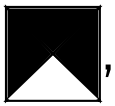
The Order-k Corridor Tiling Problem

Instance $\mathcal{K}$ is set $T = \{$  ,  , $\ldots,$  $\}$ of tiles

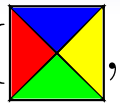
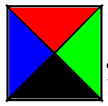
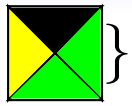
The Order-k Corridor Tiling Problem

Instance $\mathcal{K}$ is set $T = \{$  ,  , $\ldots,$  $\}$ of tiles, horizontal and vertical matching relations H and V (indicated by colors)

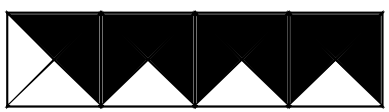
The Order-k Corridor Tiling Problem

Instance $\mathcal{K}$ is set $T = \{$  ,  , $\ldots$,  $\}$ of tiles, horizontal and vertical matching relations H and V (indicated by colors), and initial tiles $t_I =$  and $T_\square =$  , and final tile $t_F =$ 

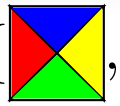
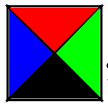
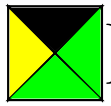
The Order-k Corridor Tiling Problem

Instance $\mathcal{K}$ is set $T = \{$  ,  , $\ldots,$  $\}$ of tiles, horizontal and vertical matching relations H and V (indicated by colors), and initial tiles $t_I =$  and $T_\square =$  , and final tile $t_F =$ 

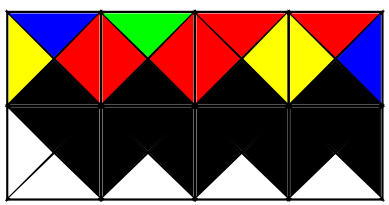
Start with initial 2^n_k wide row,



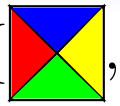
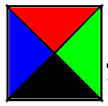
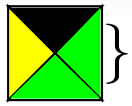
The Order-k Corridor Tiling Problem

Instance $\mathcal{K}$ is set $T = \{$  ,  , $\ldots$,  $\}$ of tiles, horizontal and vertical matching relations H and V (indicated by colors), and initial tiles $t_I =$  and $T_\square =$  , and final tile $t_F =$ 

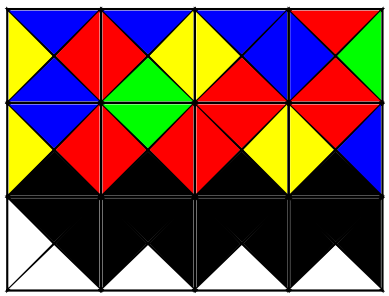
Start with initial 2^n_k wide row, find matching rows,



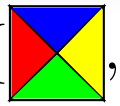
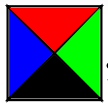
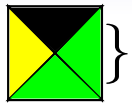
The Order-k Corridor Tiling Problem

Instance $\mathcal{K}$ is set $T = \{$  $,$  $, \ldots,$  $\}$ of tiles, horizontal and vertical matching relations H and V (indicated by colors), and initial tiles $t_I =$  and $T_\square =$  , and final tile $t_F =$ 

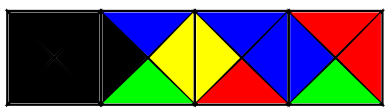
Start with initial 2^n_k wide row, find matching rows,



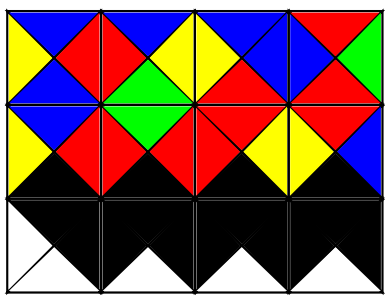
The Order-k Corridor Tiling Problem

Instance $\mathcal{K}$ is set $T = \{$  ,  , $\ldots,$  $\}$ of tiles, horizontal and vertical matching relations H and V (indicated by colors), and initial tiles $t_I =$  and $T_\square =$  , and final tile $t_F =$ 

Start with initial 2^n_k wide row, find matching rows, until a row ends with final tile



$\ldots$ $\ldots$



For $k \geq 0$, the order-k corridor tiling problem is k-EXPSPACE hard (van Emde Boas '97)

Jones' Encoding of Large Numbers

Observation: Numbers up to 2^n_{k+1} have at most 2^n_k bits in binary

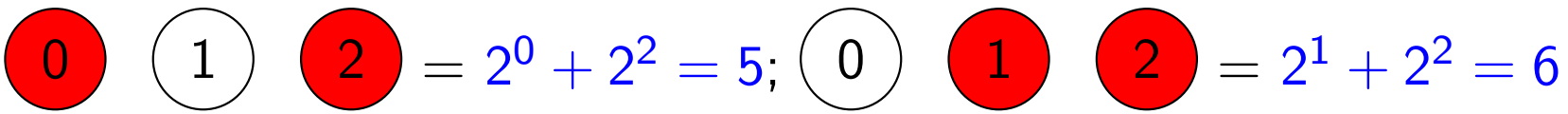
Fix an LTS with states $\mathcal{S} = \{0, \ldots, n-1\}$

Jones' Encoding of Large Numbers

Observation: Numbers up to 2^n_{k+1} have at most 2^n_k bits in binary

Fix an LTS with states $\mathcal{S} = \{0, \ldots, n-1\}$

Idea (Jones 01): Use states as bits and encode numbers as higher-order functions:

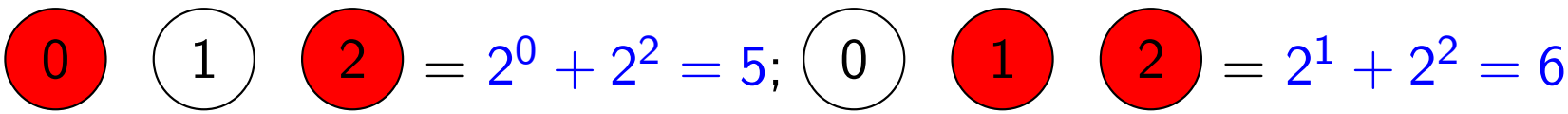


Jones' Encoding of Large Numbers

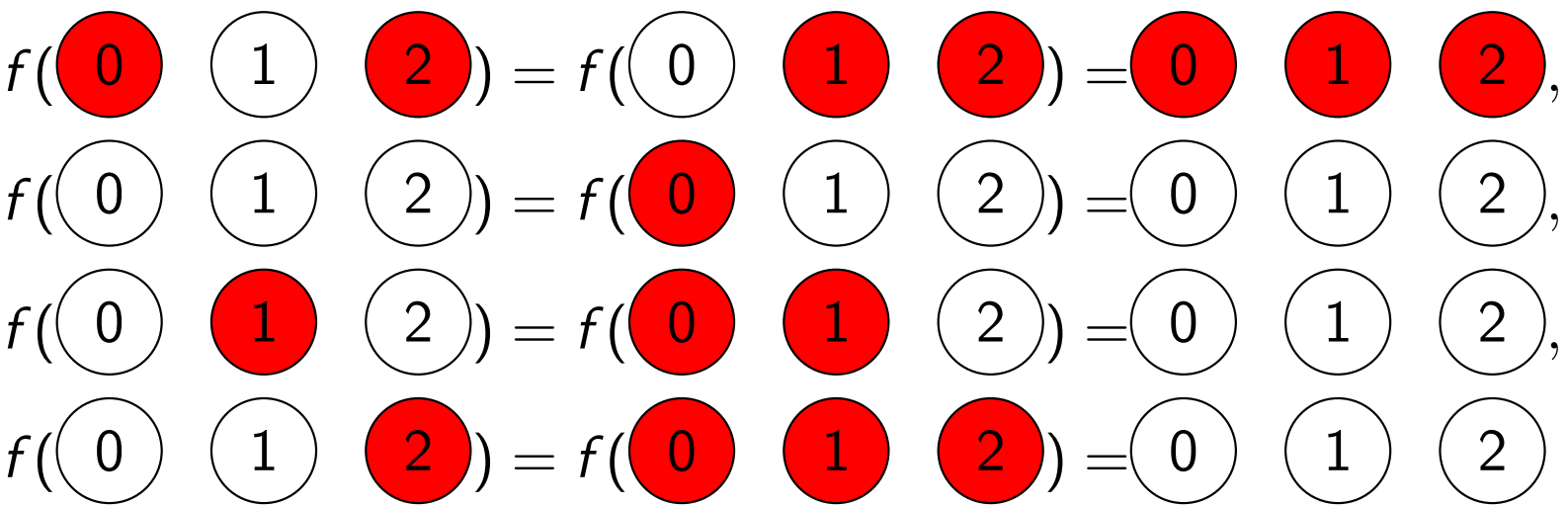
Observation: Numbers up to 2^n_{k+1} have at most 2^n_k bits in binary

Fix an LTS with states $\mathcal{S} = \{0, \ldots, n-1\}$

Idea (Jones 01): Use states as bits and encode numbers as higher-order functions:



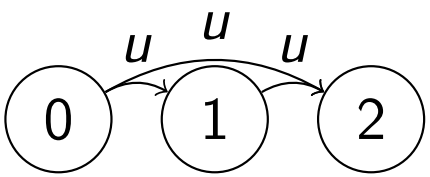
f with



$= 2^5 + 2^6 = 32 + 64 = 96$

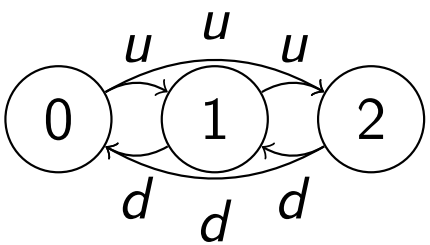
Working with Jones Encodings

Add transitions u



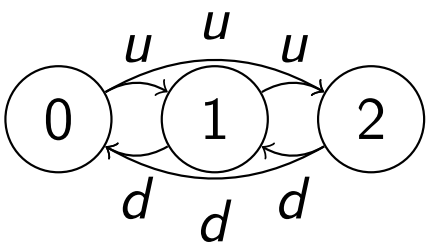
Working with Jones Encodings

Add transitions u, d and e $= \mathcal{S} \times \mathcal{S}$.



Working with Jones Encodings

Add transitions u, d and $\mathsf{e} = \mathcal{S} \times \mathcal{S}$.

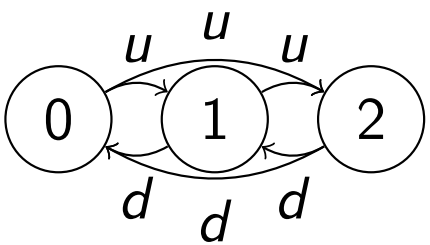


$\rightarrow$ Compare numbers: m_1 bigger than m_2 if ex. a bit set in m_1 not set in m_2, and all higher bits are set in m_1 if they are set in m_2:

$$\mathsf{gt}_0 = \lambda(m_1, m_2 : \tau_0).\ \langle \mathsf{e} \rangle \big(m_2 \wedge \neg m_1 \wedge [\mathsf{u}](m_1 \Rightarrow m_2) \big)$$

Working with Jones Encodings

Add transitions u, d and $\mathsf{e} = \mathcal{S} \times \mathcal{S}$.



$\rightarrow$ Compare numbers: m_1 bigger than m_2 if ex. a bit set in m_1 not set in m_2, and all higher bits are set in m_1 if they are set in m_2:

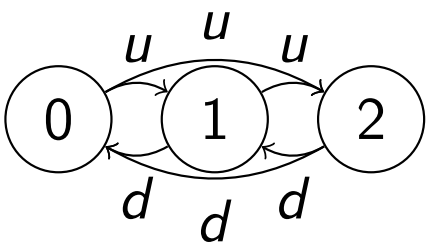
$$\mathsf{gt}_0 = \lambda(m_1, m_2 : \tau_0).\ \langle \mathsf{e} \rangle \big(m_2 \wedge \neg m_1 \wedge [\mathsf{u}](m_1 \Rightarrow m_2) \big)$$

$\rightarrow$ Increment a number: Set a bit iff it was set and a lower bit was not set, or if it was not set and all lower bits were set:

$$\mathsf{inc}_0 = \lambda(m : \mathsf{Pr}).\ \text{if } m \text{ then } (\langle \mathsf{d} \rangle \neg m) \text{ else } ([\mathsf{d}] m)$$

Working with Jones Encodings

Add transitions u, d and $\mathsf{e} = \mathcal{S} \times \mathcal{S}$.



$\rightarrow$ Compare numbers: m_1 bigger than m_2 if ex. a bit set in m_1 not set in m_2, and all higher bits are set in m_1 if they are set in m_2:

$$\mathsf{gt}_0 = \lambda(m_1, m_2 : \tau_0).\ \langle \mathsf{e} \rangle \big(m_2 \wedge \neg m_1 \wedge [\mathsf{u}](m_1 \Rightarrow m_2) \big)$$

$\rightarrow$ Increment a number: Set a bit iff it was set and a lower bit was not set, or if it was not set and all lower bits were set:

$$\mathsf{inc}_0 = \lambda(m : \mathsf{Pr}).\ \mathsf{if}\ m\ \mathsf{then}\ (\langle \mathsf{d} \rangle \neg m)\ \mathsf{else}\ ([\mathsf{d}]m)$$

$\rightarrow$ Find a number matching a predicate:

$$\mathsf{find}_k = \lambda(p : \tau_{k+1}).$$
$$\Big(\big(\mu(F : \tau_k \to \mathsf{Pr})\,.\ \lambda(m : \tau_k)\,.\ ([\mathsf{e}](p\ m)) \vee F\ (\mathsf{inc}_k\ m) \big)\ \mathsf{zero}_k \Big)$$

Encoding the Corridor Tiling Problem

Fix corridor tiling problem $\mathcal{K} = (T, H, V, t_I, t_\square, t_F)$ and extend LTS from previous slides with propositions $p_I = \{0\}, p_\square = \{|T| - 2\}, p_F = \{|T| - 1\}$ and transitions $\mathsf{h} = \{(i,j) \colon (t_i, t_j) \in H\}$, $\mathsf{v} = \{(i,j) \colon (t_i, t_j) \in V\}$.

States double as digits and tiles!

Encoding the Corridor Tiling Problem

Fix corridor tiling problem $\mathcal{K} = (T, H, V, t_I, t_\square, t_F)$ and extend LTS from previous slides with propositions $p_I = \{0\}, p_\square = \{|T| - 2\}, p_F = \{|T| - 1\}$ and transitions $\mathsf{h} = \{(i,j) \colon (t_i, t_j) \in H\}$, $\mathsf{v} = \{(i,j) \colon (t_i, t_j) \in V\}$.

States double as digits and tiles!

Encoding the Corridor Tiling Problem

Fix corridor tiling problem $\mathcal{K} = (T, H, V, t_I, t_\square, t_F)$ and extend LTS from previous slides with propositions $p_I = \{0\}, p_\square = \{|T| - 2\}, p_F = \{|T| - 1\}$ and transitions $\mathsf{h} = \{(i,j) : (t_i, t_j) \in H\}$, $\mathsf{v} = \{(i,j) : (t_i, t_j) \in V\}$.

States double as digits and tiles!

Can encode a row of tiles of width 2^n_k as function of order 2^n_{k+1} that sends a column number to the singleton set of its tile

Encoding the Corridor Tiling Problem

Fix corridor tiling problem $\mathcal{K} = (T, H, V, t_I, t_\square, t_F)$ and extend LTS from previous slides with propositions $p_I = \{0\}, p_\square = \{|T| - 2\}, p_F = \{|T| - 1\}$ and transitions $\mathsf{h} = \{(i,j) : (t_i, t_j) \in H\}$, $\mathsf{v} = \{(i,j) : (t_i, t_j) \in V\}$.

States double as digits and tiles!

Can encode a row of tiles of width 2^n_k as function of order 2^n_{k+1} that sends a column number to the singleton set of its tile

Check for final row:

$\mathsf{isFinal} = \lambda(r : \tau_k).\ [\mathsf{e}]\big((r\ \mathsf{zero}_{k-1}) \Rightarrow p_F\big)$

The Lower Bound

Formula that encodes problem:
$\big(\mu(P\colon \tau_{k+1}).\, \lambda(r_1\colon \tau_k).(\mathsf{isFinal}\ r_1) \vee (\mathsf{exists\_succ}\ r_1\ P)\big)\ \mathsf{init}$

where $\mathsf{exists\_succ} =$
$\lambda(r_1\colon \tau_k, p\colon \tau_{k+1}).\ \mathsf{find}\ \big(\lambda(r_2\colon \tau_k).\ (\mathsf{horiz}\ r_2) \wedge (\mathsf{vert}\ r_1\ r_2) \wedge (p\ r_2).$
and horiz_k verifies horizontal matching in row, and
veriz_k verifies that two rows match

The Lower Bound

Formula that encodes problem:
$\big(\mu(P\colon \tau_{k+1}).\, \lambda(r_1\colon \tau_k).(\mathsf{isFinal}\ r_1) \vee (\mathsf{exists\_succ}\ r_1\ P)\big)\ \mathsf{init}$

where $\mathsf{exists\_succ} =$
$\lambda(r_1\colon \tau_k, p\colon \tau_{k+1}).\ \mathsf{find}\ \big(\lambda(r_2\colon \tau_k).\ (\mathsf{horiz}\ r_2) \wedge (\mathsf{vert}\ r_1\ r_2) \wedge (p\ r_2).$
and horiz_k verifies horizontal matching in row, and
veriz_k verifies that two rows match

Not tail-recursive

The Lower Bound

Formula that encodes problem:
$\big(\mu(P\colon \tau_{k+1}).\, \lambda(r_1\colon \tau_k).(\text{isFinal } r_1) \vee (\text{exists\_succ } r_1\ P)\big)\ \text{init}$

where $\text{exists\_succ} =$
$\lambda(r_1\colon \tau_k, p\colon \tau_{k+1}).\ \text{find}\ \big(\lambda(r_2\colon \tau_k).\ (\text{horiz } r_2) \wedge (\text{vert } r_1\ r_2) \wedge (p\ r_2).$
and horiz_k verifies horizontal matching in row, and
veriz_k verifies that two rows match

Not tail-recursive, but can β-reduce:

$$\Big(\mu(P\colon \tau_{k+1}).\, \lambda(r_1\colon \tau_k).\quad (\dots) \vee \big(\mu(F\colon \tau_{k+1}).\lambda(r_2\colon \tau_k)\ ((\dots) \wedge (P\ r_2)) \vee (F\ (\text{inc } r_2))\big)\ r_1\Big)\ \text{init}$$

Summary

Order-$k+1$ tail-recursive HFL captures k-EXPSPACE
(as opposed to $k+1$-EXPTIME for full $k+1$-HFL)

Summary

Order-$k+1$ tail-recursive HFL captures k-EXPSPACE
(as opposed to $k+1$-EXPTIME for full $k+1$-HFL)

Remark: Instead of unlimited $\vee, \langle\rangle$, can also use unlimited $\wedge, []$,
but restricted $\vee, \langle\rangle$

Summary

Order-$k+1$ tail-recursive HFL captures k-EXPSPACE
(as opposed to $k+1$-EXPTIME for full $k+1$-HFL)

Remark: Instead of unlimited $\vee, \langle\rangle$, can also use unlimited $\wedge, []$, but restricted $\vee, \langle\rangle$; can even mix both if "interface" right, i.e., free fp vars separated

Summary

Order-$k+1$ tail-recursive HFL captures k-EXPSPACE
(as opposed to $k+1$-EXPTIME for full $k+1$-HFL)

Remark: Instead of unlimited $\vee, \langle\rangle$, can also use unlimited $\wedge, []$, but restricted $\vee, \langle\rangle$; can even mix both if "interface" right, i.e., free fp vars separated

Further research: Look whether/how much tail-recursion requirement can be removed from low-type fixpoints (probably not entirely)

Summary

Order-$k+1$ tail-recursive HFL captures k-EXPSPACE
(as opposed to $k+1$-EXPTIME for full $k+1$-HFL)

Remark: Instead of unlimited $\vee, \langle\rangle$, can also use unlimited $\wedge, []$, but restricted $\vee, \langle\rangle$; can even mix both if "interface" right, i.e., free fp vars separated

Further research: Look whether/how much tail-recursion requirement can be removed from low-type fixpoints (probably not entirely)

Thanks!