Computational interpretation of Cohen forcing

Lionel Rieg

ENSIIE

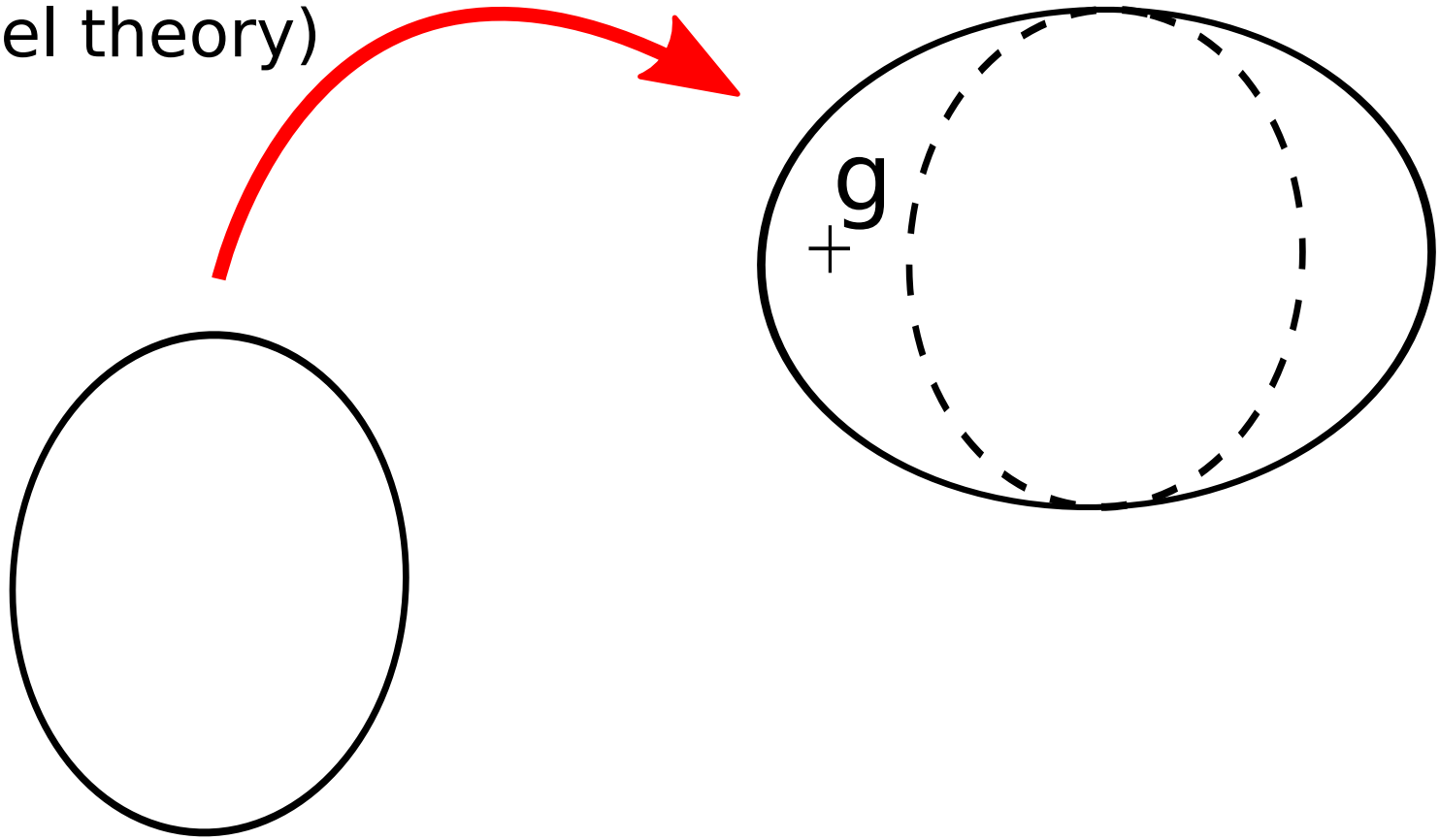
November 18$^{\text{th}}$, 2013

The question

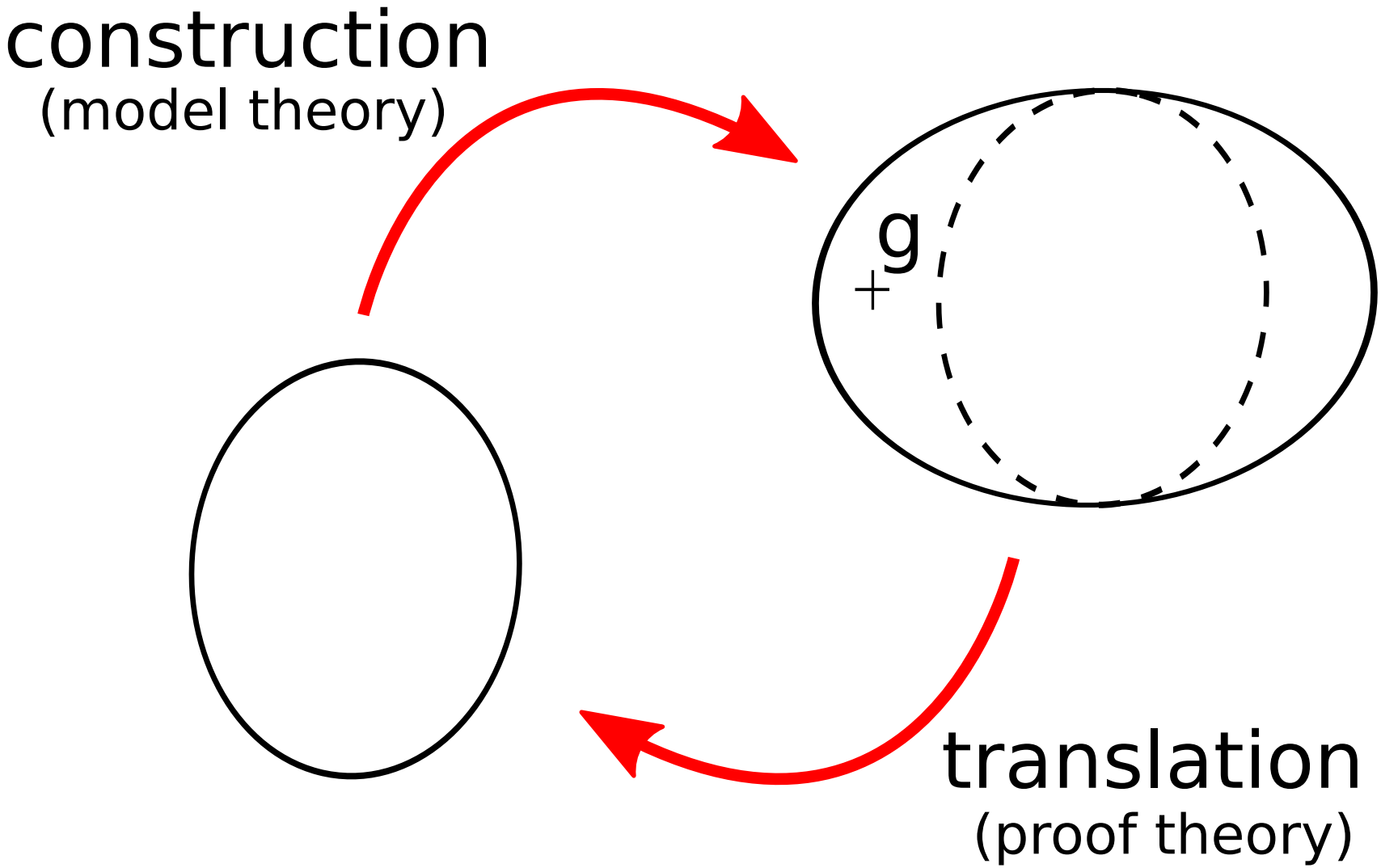
Logic	Programs
$\neg\neg$-translation	CPS translation
$\rightsquigarrow$ formula $\bot$	$\rightsquigarrow$ return type
Forcing	
$\rightsquigarrow$ forcing conditions	???
$\rightsquigarrow$ forcing transformation	

Forcing in one drawing

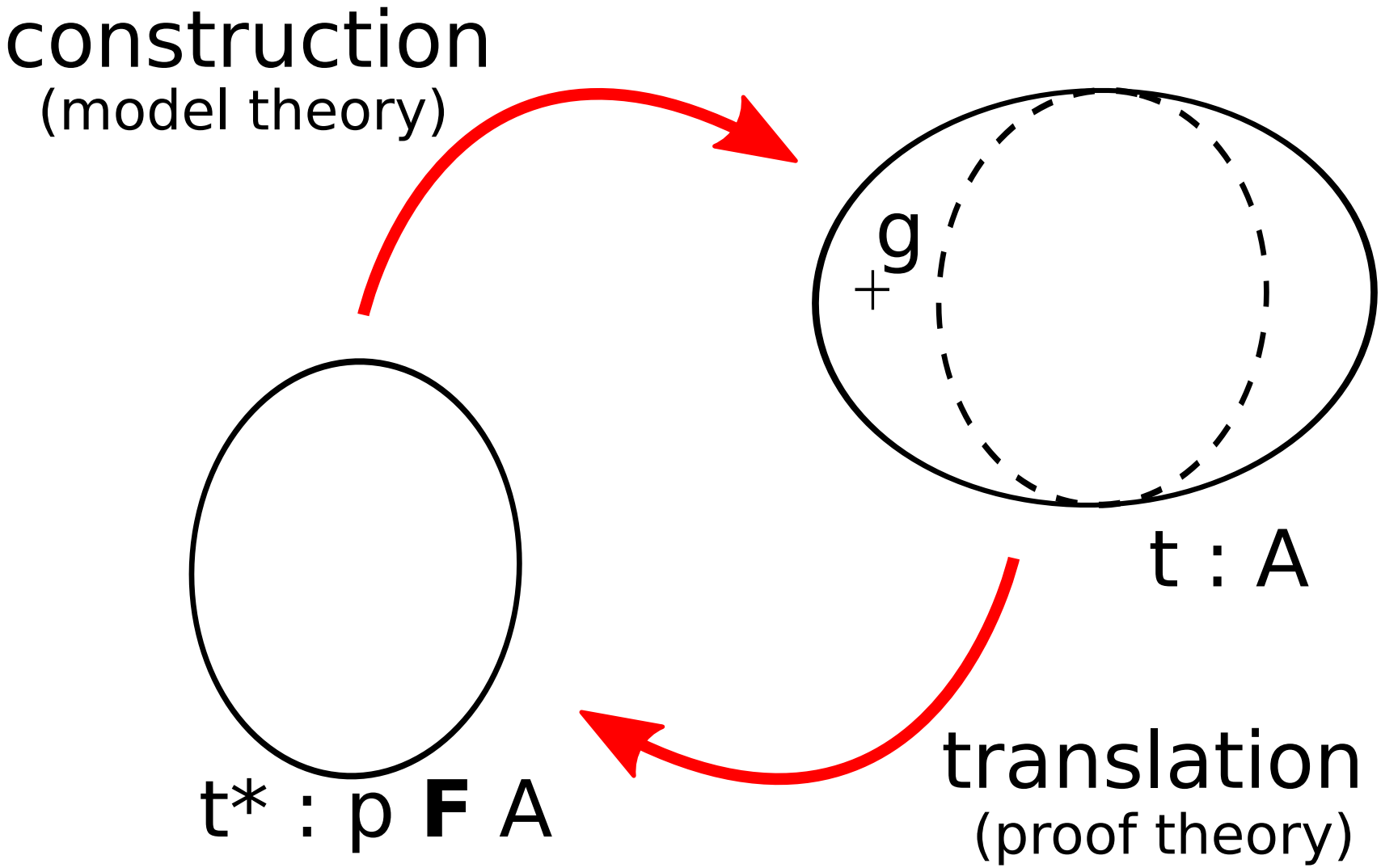
construction
(model theory)



Forcing in one drawing



Forcing in one drawing



PAω^+: syntax

Sorts

$$\tau, \sigma \quad := \quad \iota \quad | \quad o \quad | \quad \tau \to \sigma$$

Expressions

$$M, N, A, B \quad := \quad x^\tau \quad | \quad \lambda x^\tau.\, M \quad | \quad M\, N \qquad \lambda\text{-calculus}$$
$$| \quad 0 \quad | \quad S \quad | \quad \mathrm{rec}_\tau \qquad \text{arithmetic}$$
$$| \quad A \Rightarrow B \quad | \quad \forall x^\tau.\, A \qquad \text{minimal logic}$$

Proof-terms

$$t, u \quad := \quad x \quad | \quad \lambda x.\, t \quad | \quad t\, u \quad | \quad \mathrm{callcc}$$

PAω^+: syntax

Sorts

$$\tau, \sigma \quad := \quad \iota \quad | \quad o \quad | \quad \tau \to \sigma$$

Expressions

$$\begin{array}{rcll} M, N, A, B & := & x^\tau \quad | \quad \lambda x^\tau.\, M \quad | \quad M\,N & \lambda\text{-calculus} \\ & | & 0 \quad | \quad S \quad | \quad \mathrm{rec}_\tau & \text{arithmetic} \\ & | & A \Rightarrow B \quad | \quad \forall x^\tau.\, A & \text{minimal logic} \end{array}$$

Proof-terms

$$t, u \quad := \quad x \quad | \quad \lambda x.\, t \quad | \quad t\,u \quad | \quad \mathrm{callcc}$$

Other connectives ($\bot$, $\neg$, $\wedge$, $\vee$, $\exists$, $=$): 2^{nd}-order encodings

Notations: $x \in P := P(x)$ $\qquad$ $\forall x \in P.\, A := \forall x.\, x \in P \Rightarrow A$

$\exists x \in P.\, A := \exists x.\, x \in P \wedge A$

PAω^+: congruence

Reflexivity, symmetry and transitivity

$$\frac{}{M \approx M} \qquad \frac{M \approx N}{N \approx M} \qquad \frac{M \approx N \qquad N \approx P}{M \approx P}$$

Context closure

$$\frac{M \approx N}{\lambda x.\, M \approx \lambda x.\, N} \qquad \frac{M \approx N \qquad P \approx Q}{M\,P \approx N\,Q}$$

$$\frac{A \approx B}{\forall x^\tau.\, A \approx \forall x^\tau.\, B} \qquad \frac{A \approx B \qquad C \approx D}{A \Rightarrow C \approx B \Rightarrow D}$$

$\beta\eta\iota$-conversion

$$\frac{}{(\lambda x^\tau.\, M)\, N^\tau \approx M[N^\tau/x^\tau]} \qquad \frac{}{\lambda x.\, M\,x \approx M}\ x \notin FV(M)$$

$$\frac{}{\mathrm{rec}_\tau\, M\, N\, 0 \approx M} \qquad \frac{}{\mathrm{rec}_\tau\, M\, N (S\, P) \approx N\, P\, (\mathrm{rec}_\tau\, M\, N\, P)}$$

Semantically equivalent propositions

$$\frac{}{\forall x^\tau \forall y^\sigma.\, A \approx \forall y^\sigma \forall x^\tau.\, A} \qquad \frac{}{\forall x^\tau.\, A \approx A}\ x \notin FV(A)$$

$$\frac{}{A \Rightarrow \forall x^\tau.\, B \approx_{\mathcal{E}} \forall x^\tau.\, A \Rightarrow B}\ x \notin FV(A)$$

PAω^+: proof system

$$\text{Axiom}\ \frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{}{\Gamma \vdash \text{callcc} : ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}\ \text{Peirce}$$

$$\text{Congruence}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash t : A'}\ A \approx A'$$

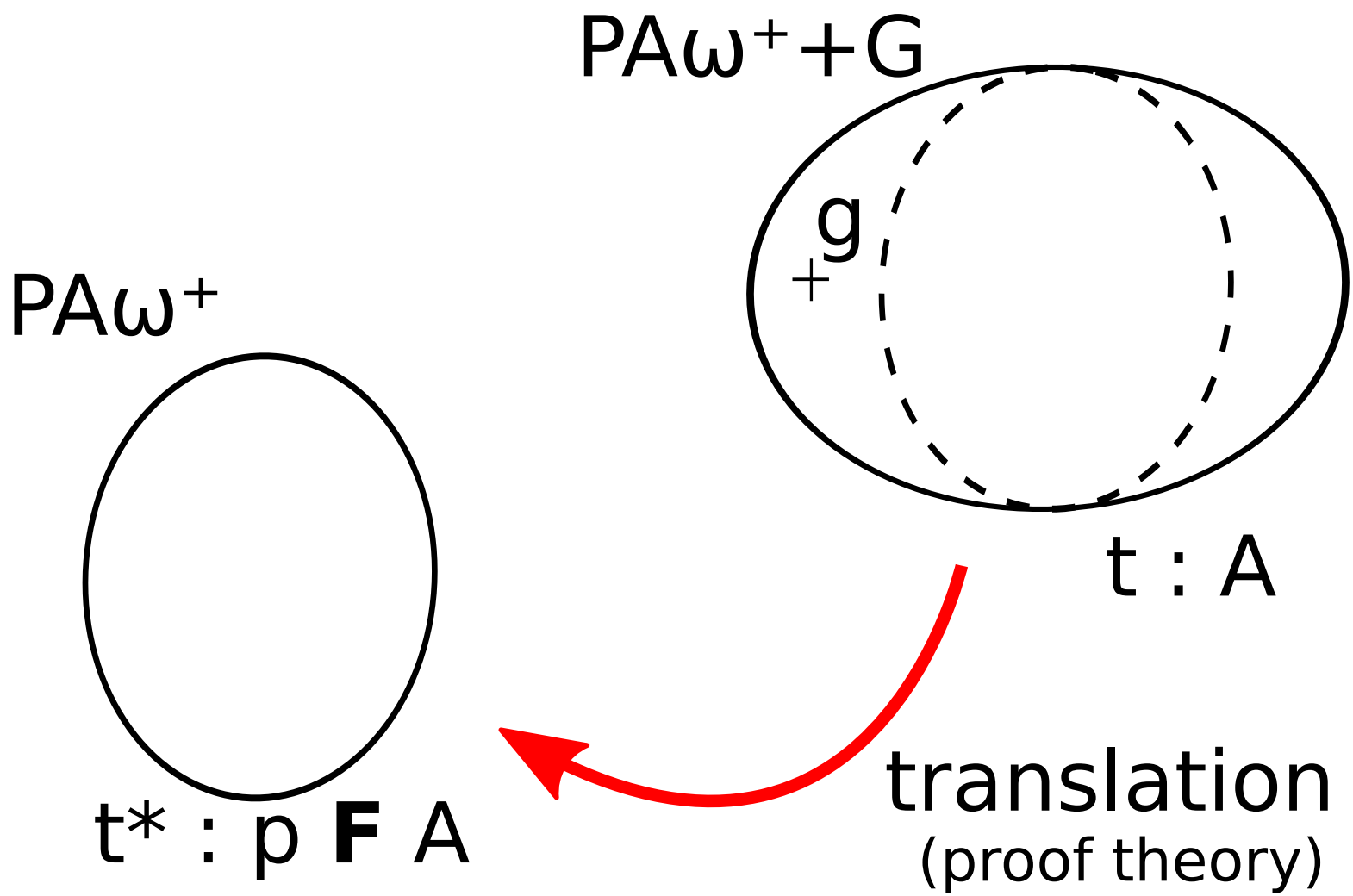
$$\Rightarrow_i \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.\, t : A \Rightarrow B} \qquad \frac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}\ \Rightarrow_e$$

$$\forall_i \frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall x^\tau.\, A}\ x \notin FV(\Gamma, \mathcal{E}) \qquad \frac{\Gamma \vdash t : \forall x^\tau.\, A}{\Gamma \vdash t : A[N^\tau / x^\tau]}\ \forall_e$$

Classical realizability semantics

- The KAM (Krivine's Abstract Machine)
  Stack machine for λ-calculus + callcc
- Realizability interpretation
  - Based on a pole $\perp\!\!\!\perp$ (set of processes of the KAM)
  - Propositions interpreted by stacks (refutations)
  - Realizers defined by orthogonality: $|A| := [\![A]\!]^{\perp\!\!\!\perp}$
- Results:
  - Adequacy: $\vdash t : A$ implies $t \Vdash A$
  - Logical consistency: when $\perp\!\!\!\perp = \varnothing$, Tarski model
  - Simple methods to extract witnesses for Σ^0_1 formulas

Forcing: overall idea



Forcing: input

Definition (Forcing structure)

A forcing structure is given by

- a sort κ of forcing conditions
- a predicate $C^{\kappa \to o}$ of well-formed conditions ($p \in C$ written $C[p]$)
- a product operation $\cdot$ on forcing conditions
- a maximal condition 1
- a bunch of proof terms $\alpha_0, \ldots, \alpha_8$

$G =$ generic filter on the set of well-formed forcing conditions
$\quad =$ "approximations of g"

$g = \bigcup G$

Forcing: output

3 translations $(\_)^*$:

Forcing: output

3 translations $(\_)^*$:

- on kinds:

$$\iota^* := \iota \qquad o^* := \kappa \to o \qquad (\sigma \to \tau)^* := \sigma^* \to \tau^*$$

Forcing: output

3 translations $(\_)^*$:

- on kinds:

$$\iota^* := \iota \qquad o^* := \kappa \to o \qquad (\sigma \to \tau)^* := \sigma^* \to \tau^*$$

- on expressions:
  - $(A \Rightarrow B)^* \, p := \forall q \forall r. \, p \doteq q \cdot r \mapsto (\forall s. \, C[q \cdot s] \Rightarrow A^* \, s) \Rightarrow B^* \, r$
  - merely propagates through other constructions

Forcing: output

3 translations $(\_)^*$:

- on kinds:

$$\iota^* := \iota \qquad o^* := \kappa \to o \qquad (\sigma \to \tau)^* := \sigma^* \to \tau^*$$

- on expressions:
  - $(A \Rightarrow B)^* \, p := \forall q \forall r. \, p \doteq q \cdot r \mapsto (\forall s. \, C[q \cdot s] \Rightarrow A^* \, s) \Rightarrow B^* \, r$
  - merely propagates through other constructions

The forcing transformation: $p \Vdash A := \forall r. \, C[p \cdot r] \Rightarrow A^* \, r$

Forcing: output

3 translations $(\_)^*$:

- on kinds:

$$\iota^* := \iota \qquad o^* := \kappa \to o \qquad (\sigma \to \tau)^* := \sigma^* \to \tau^*$$

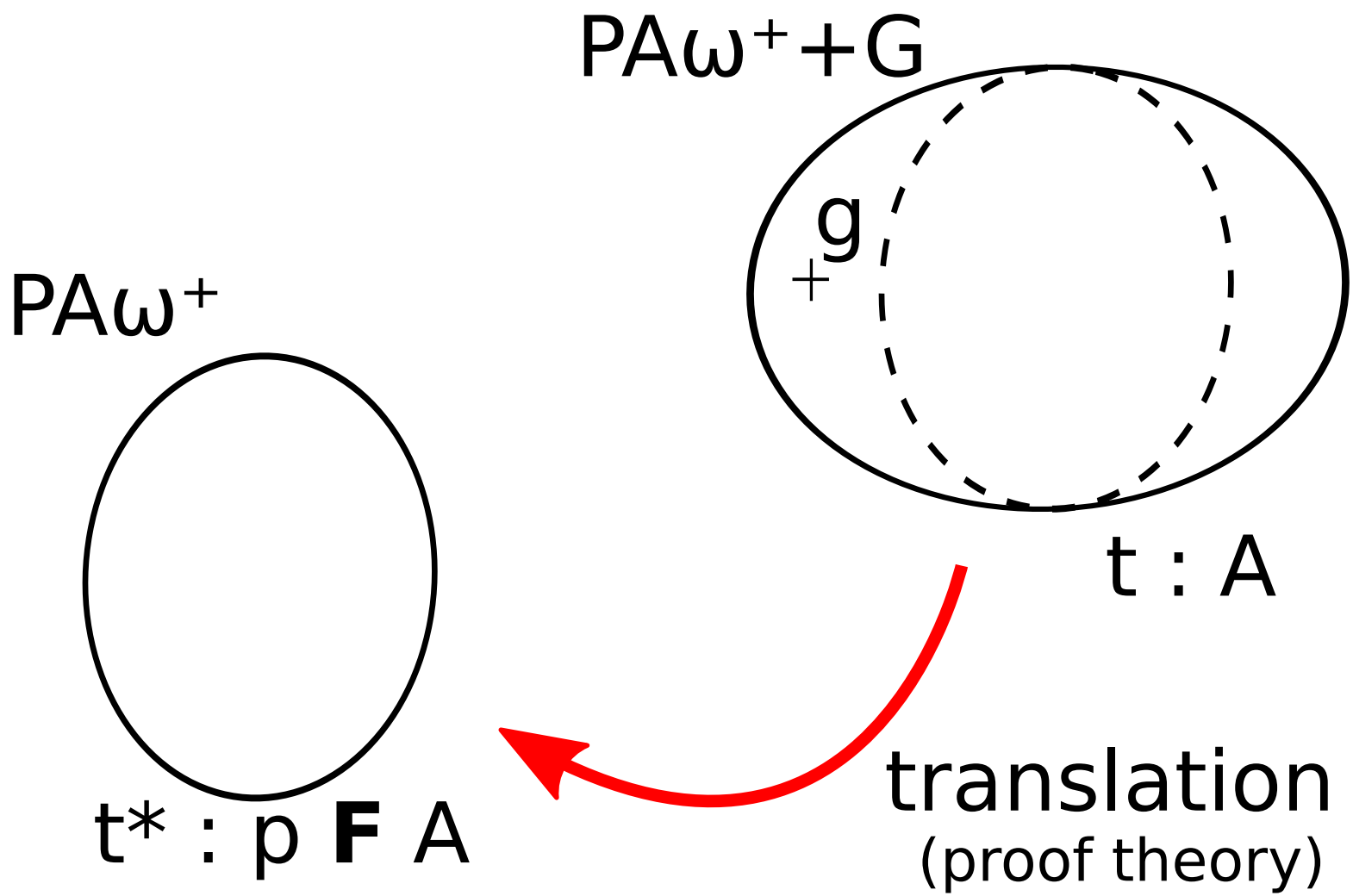
- on expressions:
  - $(A \Rightarrow B)^* \, p := \forall q \forall r. \, p \doteq q \cdot r \mapsto (\forall s. \, C[q \cdot s] \Rightarrow A^* \, s) \Rightarrow B^* \, r$
  - merely propagates through other constructions

  The forcing transformation: $p \Vdash A := \forall r. \, C[p \cdot r] \Rightarrow A^* \, r$

- on proof terms:

$$\begin{array}{lll} x^* & := x & \\ (t\,u)^* & := \gamma_3 \, t^* \, u^* & \\ (\lambda x. \, t)^* & := \gamma_1 (\lambda x. \, t^* [(\beta_3 y)/y][(\beta_4 x)/x]) & y \neq x \\ \mathsf{callcc}^* & := \lambda c x. \, \mathsf{callcc} \, (\lambda k. \, x \, (\alpha_{14} \, c) \, (\gamma_4 \, k)) & \end{array}$$

Forcing: overall idea



Forcing: extension to the generic filter

Restriction: C is invariant by forcing (arithmetical)

Forcing: extension to the generic filter

Restriction: C is invariant by forcing (arithmetical)

$$\begin{array}{ccc} \mathrm{PA}\omega^+ + G & \longrightarrow & \boxed{\text{Forcing translation}} \longrightarrow \quad \mathrm{PA}\omega^+ \\ A & & p \Vdash A \\ t : A & & t^* : p \Vdash A \\ q \in G & & ?? \end{array}$$

Forcing: extension to the generic filter

Restriction: C is invariant by forcing (arithmetical)

$$\begin{array}{ccc} \mathrm{PA}\omega^+ + G & \longrightarrow \boxed{\text{Forcing translation}} \longrightarrow & \mathrm{PA}\omega^+ \\ A & & p \Vdash A \\ t : A & & t^* : p \Vdash A \\ q \in G & & p \leqslant q \end{array}$$

$$p \Vdash q \in G \quad \equiv \quad p \leqslant q \quad := \quad \forall r.\, C[p \cdot r] \Rightarrow C[q \cdot r]$$

Forcing: extension to the generic filter

Restriction: C is invariant by forcing (arithmetical)

$$\begin{array}{ccc} \mathrm{PA}\omega^+ + G & \longrightarrow & \boxed{\text{Forcing translation}} \longrightarrow \quad \mathrm{PA}\omega^+ \\ A & & p \Vdash A \\ t : A & & t^* : p \Vdash A \\ q \in G & & p \leqslant q \end{array}$$

$$p \Vdash q \in G \quad \equiv \quad p \leqslant q \quad := \quad \forall r.\, C[p \cdot r] \Rightarrow C[q \cdot r]$$

Nice properties of G in the forcing universe:

- non empty $\quad 1 \in G$
- subset of C $\quad \forall p \in G.\, C[p]$
- filter $\quad \forall p \forall q.\, (p \cdot q) \in G \Rightarrow p \in G$
  $\quad \forall p \in G. \forall q \in G.\, (p \cdot q) \in G$
- genericity $\quad \ldots$

We need to prove that they are forced

Forcing usage : the big picture

We want to prove $\dfrac{A_1 \quad \ldots \quad A_n}{A}$.

Base universe

Forcing universe

Forcing usage : the big picture

We want to prove $\dfrac{A_1 \quad \ldots \quad A_n}{A}$.

Base universe

1. Build the forcing structure

Forcing universe

Forcing usage : the big picture

We want to prove $\dfrac{A_1 \quad \ldots \quad A_n}{A}$.

Base universe

1. Build the forcing structure
2. Assume the premises $x_1 \ldots x_n$

Forcing universe

Forcing usage : the big picture

We want to prove $\dfrac{A_1 \quad \ldots \quad A_n}{A}$.

Base universe

1. Build the forcing structure
2. Assume the premises $x_1 \ldots x_n$

Forcing universe

3. Lift the premises $x_1 \ldots x_n$

Forcing usage : the big picture

We want to prove $\dfrac{A_1 \quad \ldots \quad A_n}{A}$.

Base universe

1. Build the forcing structure
2. Assume the premises $x_1 \ldots x_n$

Forcing universe

3. Lift the premises $x_1 \ldots x_n$
4. Make the proof (using g/G) $t(x_1, \ldots, x_n) : A$

Forcing usage : the big picture

We want to prove $\dfrac{A_1 \quad \ldots \quad A_n}{A}$.

Base universe

1. Build the forcing structure
2. Assume the premises $x_1 \ldots x_n$

5. Use the forcing translation $t^*(x_1^*, \ldots, x_n^*) : 1 \Vdash A$

Forcing universe

3. Lift the premises $x_1 \ldots x_n$
4. Make the proof (using g/G) $t(x_1, \ldots, x_n) : A$

Forcing usage : the big picture

We want to prove $\dfrac{A_1 \quad \ldots \quad A_n}{A}$.

Base universe

1. Build the forcing structure
2. Assume the premises $x_1 \ldots x_n$

5. Use the forcing translation $t^*(x_1^*, \ldots, x_n^*) : 1 \Vdash A$
6. Remove forcing $w\, t^*(x_1^*, \ldots, x_n^*) : A$

Forcing universe

3. Lift the premises $x_1 \ldots x_n$
4. Make the proof (using g/G) $t(x_1, \ldots, x_n) : A$

Forcing usage : the big picture

We want to prove $\dfrac{A_1 \quad \ldots \quad A_n}{A}$.

Base universe

1. Build the forcing structure
2. Assume the premises $x_1 \ldots x_n$

5. Use the forcing translation $t^*(x_1^*, \ldots, x_n^*) : 1 \Vdash A$
6. Remove forcing $w\, t^*(x_1^*, \ldots, x_n^*) : A$
7. Extract a witness (classical realizability)

Forcing universe

3. Lift the premises $x_1 \ldots x_n$
4. Make the proof (using g/G) $t(x_1, \ldots, x_n) : A$

The KFAM: regular mode

Like the KAM

terms	t, u	$:=$	$x \mid \lambda x.\, t \mid t\, u \mid \mathsf{callcc} \mid \ldots$
environments	e	$:=$	$\varnothing \mid e, x \leftarrow c$
closures	c	$:=$	$t[e] \mid k_\pi$
stacks	π	$:=$	$\alpha \mid c \cdot \pi$
processes	p	$:=$	$c \star \pi$

The KFAM: regular mode

Like the KAM

terms	t, u	$:=$	x	$\mid$	$\lambda x.\, t$	$\mid$	$t\, u$	$\mid$ callcc $\mid$ $\ldots$
environments	e	$:=$	$\varnothing$	$\mid$	$e, x \leftarrow c$			
closures	c	$:=$	$t[e]$	$\mid$	k_π			
stacks	π	$:=$	α	$\mid$	$c \cdot \pi$			
processes	p	$:=$	$c \star \pi$					

Skip	$x[e, y \leftarrow c] \star$	π	$\succ$	$x[e] \star$	π		
Access	$x[e, x \leftarrow c] \star$	π	$\succ$	$c \star$	π		
Push	$(t\, u)[e] \star$	π	$\succ$	$t[e] \star$	$u[e] \cdot \pi$		
Grab	$(\lambda x.\, t)[e] \star$	$c \cdot \pi$	$\succ$	$t[e, x \leftarrow c] \star$	π		
Save	$\text{callcc}[e] \star$	$c \cdot \pi$	$\succ$	$c \star$	$k_\pi \cdot \pi$		
Restore	$k_{\pi'} \star$	$c \cdot \pi$	$\succ$	$c \star$	π'		

The KFAM: regular mode

Like the KAM + forcing

terms	t, u	$:=$	x	$\mid$	$\lambda x.\, t$	$\mid$	$t\, u$	$\mid$ callcc $\mid$ $\ldots$
environments	e	$:=$	$\varnothing$	$\mid$	$e, x \leftarrow c$			
closures	c	$:=$	$t[e]$	$\mid$	k_π	$\mid$	$t[e]^*$	$\mid$ k_π^*
stacks	π	$:=$	α	$\mid$	$c \cdot \pi$			
processes	p	$:=$	$c \star \pi$					

Skip	$x[e, y \leftarrow c]$	$\star$	π	$\succ$	$x[e]$	$\star$	π
Access	$x[e, x \leftarrow c]$	$\star$	π	$\succ$	c	$\star$	π
Push	$(t\, u)[e]$	$\star$	π	$\succ$	$t[e]$	$\star$	$u[e] \cdot \pi$
Grab	$(\lambda x.\, t)[e]$	$\star$	$c \cdot \pi$	$\succ$	$t[e, x \leftarrow c]$	$\star$	π
Save	callcc$[e]$	$\star$	$c \cdot \pi$	$\succ$	c	$\star$	$k_\pi \cdot \pi$
Restore	$k_{\pi'}$	$\star$	$c \cdot \pi$	$\succ$	c	$\star$	π'

The KFAM: evaluation

Skip	$x[e, y \leftarrow c]$	$\star$		π	$\succ$	$x[e]$	$\star$		π
Access	$x[e, x \leftarrow c]$	$\star$		π	$\succ$	c	$\star$		π
Push	$(t\,u)[e]$	$\star$		π	$\succ$	$t[e]$	$\star$	$u[e]\cdot$	π
Grab	$(\lambda x.\,t)[e]$	$\star$	$c\cdot$	π	$\succ$	$t[e, x \leftarrow c]$	$\star$		π
Save	$\mathrm{callcc}[e]$	$\star$	$c\cdot$	π	$\succ$	c	$\star$	$k_\pi\cdot$	π
Restore	$k_{\pi'}$	$\star$	$c\cdot$	π	$\succ$	c	$\star$		π'

$\Uparrow \qquad \Downarrow$

Skip*	$x[e, y \leftarrow c]^*$	$\star$	$f\cdot$	π	$\succ$	$x[e]^*$	$\star$	$\alpha_9\ f\cdot$	π
Access*	$x[e, x \leftarrow c]^*$	$\star$	$f\cdot$	π	$\succ$	c	$\star$	$\alpha_{10}\ f\cdot$	π
Push*	$(t\,u)[e]^*$	$\star$	$f\cdot$	π	$\succ$	$t[e]^*$	$\star$	$\alpha_{11}\ f\cdot u[e]^*\cdot$	π
Grab*	$(\lambda x.\,t)[e]^*$	$\star$	$f\cdot c\cdot$	π	$\succ$	$t[e, x \leftarrow c]^*$	$\star$	$\alpha_6\ f\cdot$	π
Save*	$\mathrm{callcc}[e]^*$	$\star$	$f\cdot c\cdot$	π	$\succ$	c	$\star$	$\alpha_{14}\ f\cdot$	$k_\pi^*\cdot\ \pi$
Restore*	$k_{\pi'}^*$	$\star$	$f\cdot c\cdot$	π	$\succ$	c	$\star$	$\alpha_{15}\ f\cdot$	π'

The KFAM: forcing mode

Skip*	$x[e, y \leftarrow c]^*$	$\star$	$f\cdot$	π	$\succ$	$x[e]^*$	$\star$	α_9	$f\cdot\ \pi$
Access*	$x[e, x \leftarrow c]^*$	$\star$	$f\cdot$	π	$\succ$	c	$\star$	α_{10}	$f\cdot\ \pi$
Push*	$(t\,u)[e]^*$	$\star$	$f\cdot$	π	$\succ$	$t[e]^*$	$\star$	α_{11}	$f\cdot u[e]^*\cdot\pi$
Grab*	$(\lambda x.\,t)[e]^*$	$\star$	$f\cdot c\cdot\pi$		$\succ$	$t[e, x \leftarrow c]^*$	$\star$	α_6	$f\cdot\ \pi$
Save*	$\mathrm{callcc}[e]^*$	$\star$	$f\cdot c\cdot\pi$		$\succ$	c	$\star$	α_{14}	$f\cdot\ k^*_\pi\cdot\pi$
Restore*	$k^*_{\pi'}$	$\star$	$f\cdot c\cdot\pi$		$\succ$	c	$\star$	α_{15}	$f\cdot\ \pi'$

$$\begin{aligned}
\alpha_9 &: C[(pq)r] \Rightarrow C[pr] \\
\alpha_{10} &: C[(pq)r] \Rightarrow C[qr] \\
\alpha_{11} &: C[pq] \Rightarrow C[p(pq)] \\
\alpha_6 &: C[p(qr)] \Rightarrow C[(pq)r] \\
\alpha_{14} &: C[p(qr)] \Rightarrow C[q(rr)] \\
\alpha_{15} &: C[p(qr)] \Rightarrow C[qp]
\end{aligned}$$

Conclusion and perspectives

- practical method for extracting proofs using forcing
- extend Curry-Howard correspondence

Logic	Programs
forcing conditions	extra-information
forcing transformation	add a memory cell
new object g	value of the memory cell
axioms on G	instructions on the memory cell

- one example (Herbrand) where forcing "=" tree library

Conclusion and perspectives

- practical method for extracting proofs using forcing
- extend Curry-Howard correspondence

Logic	Programs
forcing conditions	extra-information
forcing transformation	add a memory cell
new object g	value of the memory cell
axioms on G	instructions on the memory cell

- one example (Herbrand) where forcing "=" tree library

- Add more examples (bar recursion)
- See how real programs behave in the KFAM
- general case for G (C non invariant by forcing)

Conclusion and perspectives

- practical method for extracting proofs using forcing
- extend Curry-Howard correspondence

Logic	Programs
forcing conditions	extra-information
forcing transformation	add a memory cell
new object g	value of the memory cell
axioms on G	instructions on the memory cell

- one example (Herbrand) where forcing "=" tree library

- Add more examples (bar recursion)
- See how real programs behave in the KFAM
- general case for G (C non invariant by forcing)

Thank you

Classical realizability interpretation

Sorts

$$\begin{aligned}
[\![\iota]\!] &:= \mathbb{N} \\
[\![o]\!] &:= \mathcal{P}(\Pi) \\
[\![\sigma \to \tau]\!] &:= [\![\tau]\!]^{[\![\sigma]\!]}
\end{aligned}$$

Terms

$$\begin{aligned}
[\![x^\tau]\!]_\rho &:= \rho(x) \\
[\![\lambda x.\, M]\!]_\rho &:= v \mapsto [\![M]\!]_{\rho, x^\tau \leftarrow v} \\
[\![MN]\!]_\rho &:= [\![M]\!]_\rho\, [\![N]\!]_\rho \\
[\![0]\!]_\rho &:= 0 \\
[\![S]\!]_\rho &:= n \mapsto n+1 \\
[\![\mathrm{rec}_\tau]\!]_\rho &:= \mathrm{rec}_{[\![\tau]\!]} \\
[\![A \Rightarrow B]\!]_\rho &:= \left\{ t \cdot \pi \;\middle|\; t \in |A|_\rho \wedge \pi \in \|B\|_\rho \right\} \\
[\![\forall x^\tau . A]\!]_\rho &:= \bigcup_{v \in [\![\tau]\!]} [\![A]\!]_{\rho, x^\tau \leftarrow v} \\
[\![M \doteq_\tau N \mapsto A]\!]_\rho &:= \begin{cases} [\![A]\!]_\rho & \text{if } [\![M]\!]_\rho = [\![N]\!]_\rho \\ \varnothing & \text{otherwise} \end{cases}
\end{aligned}$$

Truth values

$$|A|_\rho := \left\{ t \in \Lambda \;\middle|\; \forall \pi \in [\![A]\!]_\rho \,.\, t \star \pi \in \perp\!\!\!\perp \right\}$$

Herbrand trees

Herbrand's theorem

If $\exists \vec{x}.\, F(\vec{x})$ is provable,
then there exists closed terms $\vec{t_1}, \ldots, \vec{t_k}$
such that $F(\vec{t_1}) \vee \ldots \vee F(\vec{t_k})$ is provable.

Herbrand trees

Herbrand's theorem (semantic version)

If $\exists \vec{x}.\, F(\vec{x})$ is true in all syntactic model,
then there exists closed terms $\vec{t_1}, \ldots, \vec{t_k}$
such that $F(\vec{t_1}) \vee \ldots \vee F(\vec{t_k})$ is provable.

Herbrand trees

Herbrand's theorem (semantic version)

If $\exists \vec{x}.\, F(\vec{x})$ is true in all syntactic model,
then there exists closed terms $\vec{t_1}, \ldots, \vec{t_k}$
such that $F(\vec{t_1}) \vee \ldots \vee F(\vec{t_k})$ is provable.

To which model correspond each witness?

Herbrand trees

Herbrand's theorem (semantic version)

If $\exists \vec{x}.\, F(\vec{x})$ is true in all syntactic model,
then there exists closed terms $\vec{t_1}, \ldots, \vec{t_k}$
such that $F(\vec{t_1}) \vee \ldots \vee F(\vec{t_k})$ is provable.

To which model correspond each witness?

Definition (Herbrand tree)

A *Herbrand tree* is a finite binary tree H such that:

- inner nodes of H are labeled with atomic formulas
  $\rightsquigarrow$ every branch of H represents a partial valuation
- Every leaf of H contains a witness $\vec{t}$ such that $\rho \models F(\vec{t})$
  (ρ total valuation extending the branch)

Usual proof of Herbrand's theorem

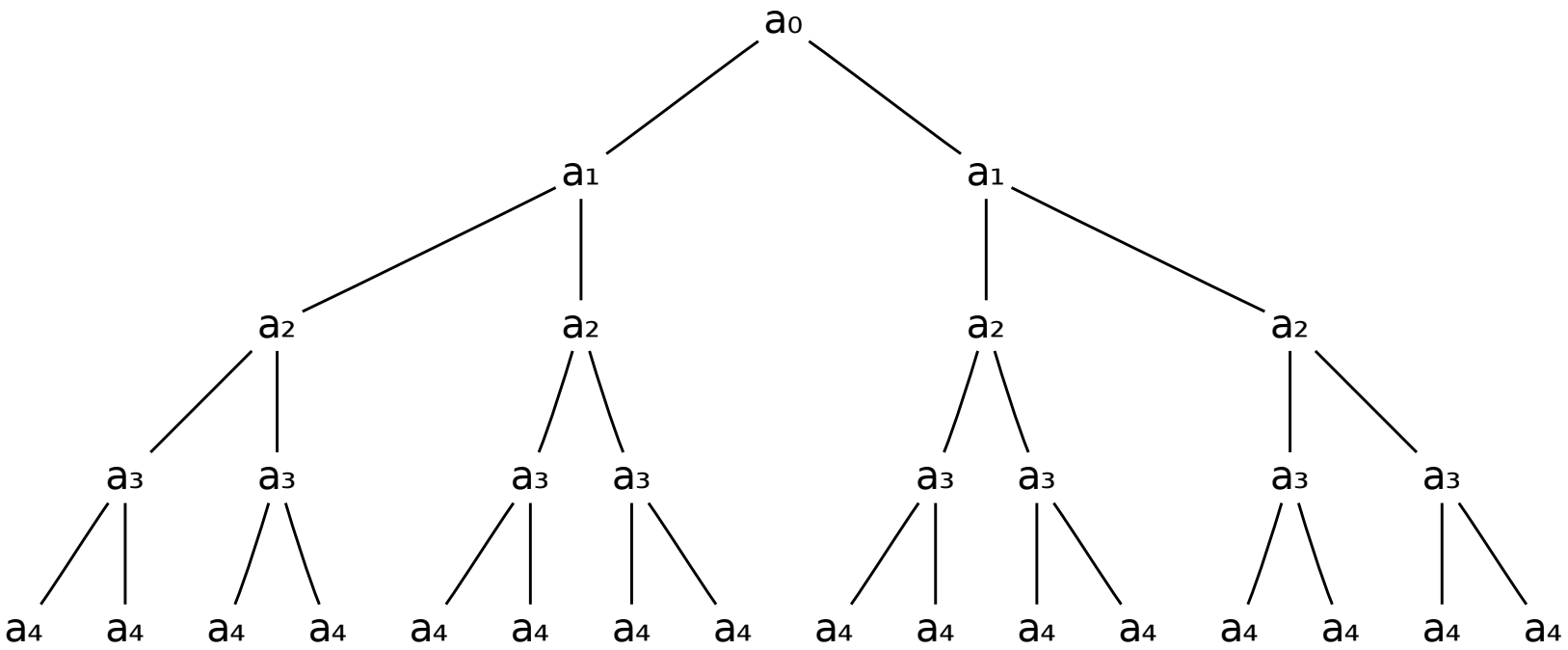
If $\exists \vec{x}.\, F(\vec{x})$ is true in all syntactic model, then there exists closed terms $\vec{t}_1, \ldots, \vec{t}_k$ such that $F(\vec{t}_1) \vee \ldots \vee F(\vec{t}_k)$ is provable.

Let us fix an enumeration $(a_i)_{i \in \mathbb{N}}$ of the atoms.

(atoms = closed atomic formulas)

Usual proof of Herbrand's theorem

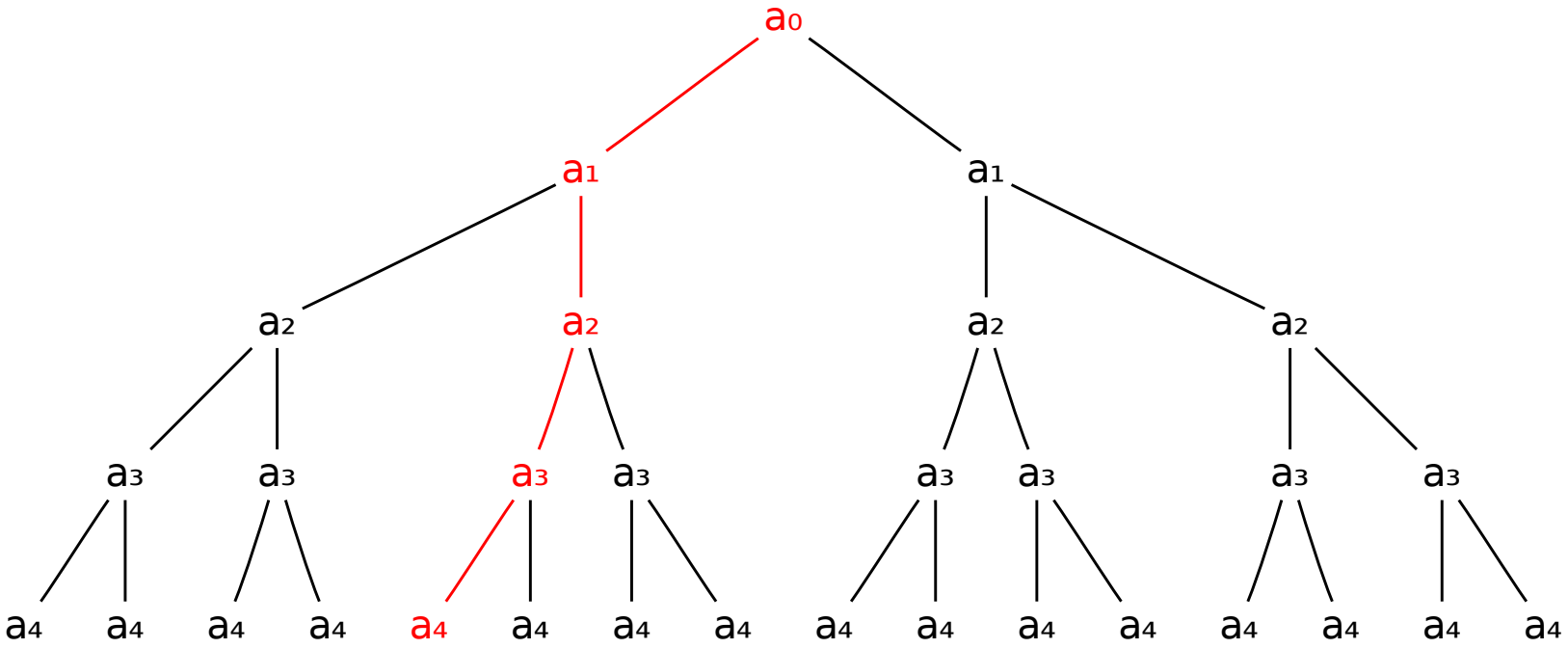
If $\exists \vec{x}.\, F(\vec{x})$ is true in all syntactic model,
then there exists closed terms $\vec{t}_1, \ldots, \vec{t}_k$
such that $F(\vec{t}_1) \vee \ldots \vee F(\vec{t}_k)$ is provable.



consider the atom-enumerating complete infinite tree

Usual proof of Herbrand's theorem

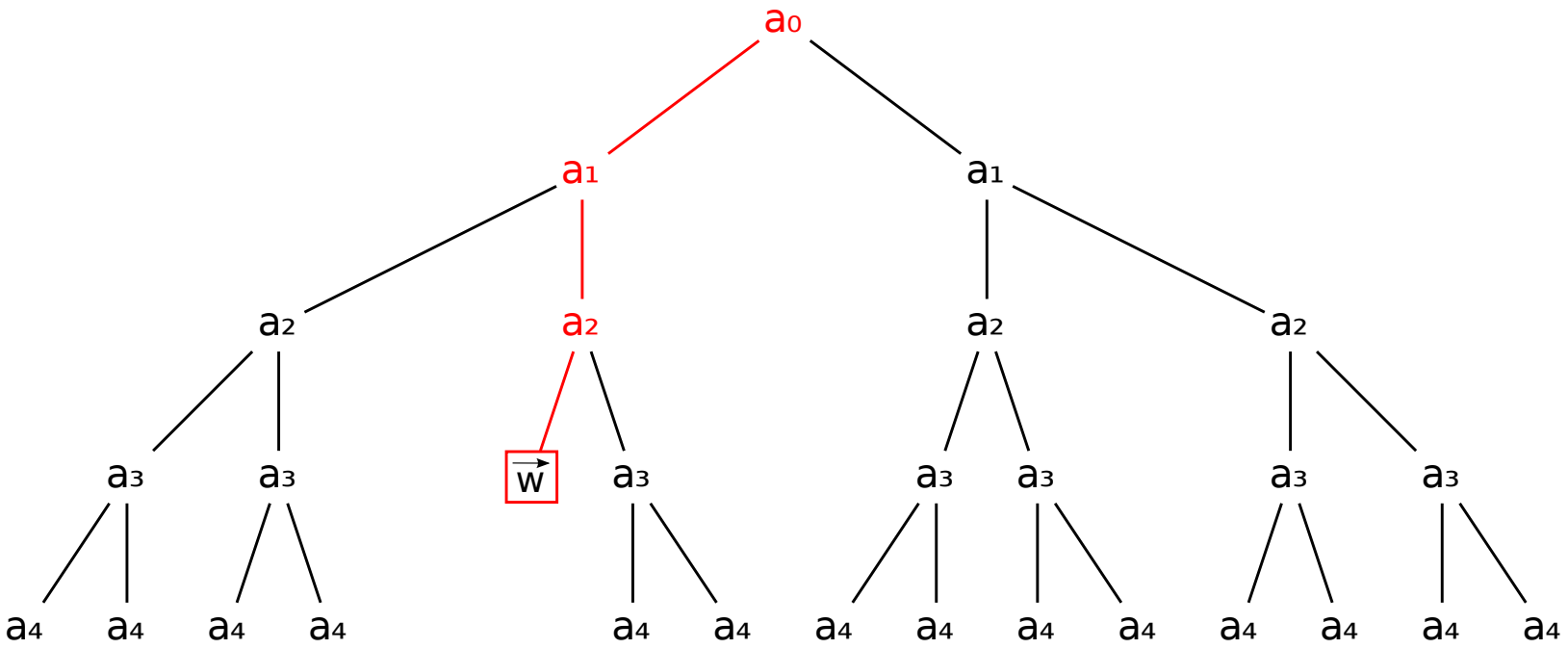
If $\exists \vec{x}.\, F(\vec{x})$ is true in all syntactic model,
then there exists closed terms $\vec{t}_1, \ldots, \vec{t}_k$
such that $F(\vec{t}_1) \vee \ldots \vee F(\vec{t}_k)$ is provable.



pick any infinite branch

Usual proof of Herbrand's theorem

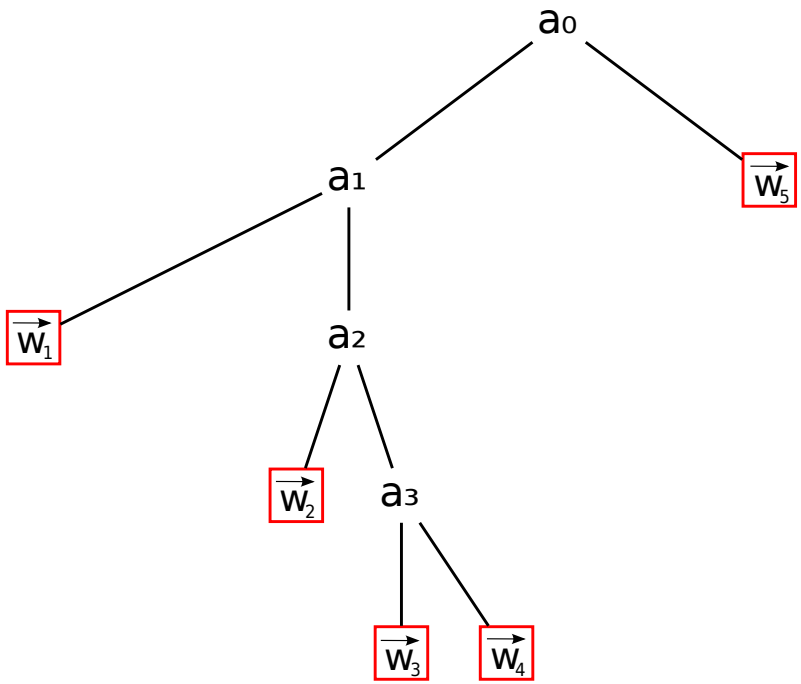
If $\exists \vec{x}.\, F(\vec{x})$ is true in all syntactic model,
then there exists closed terms $\vec{t}_1, \ldots, \vec{t}_k$
such that $F(\vec{t}_1) \vee \ldots \vee F(\vec{t}_k)$ is provable.



by hypothesis (and $F(\vec{w})$ finite), we can cut it at finite depth

Usual proof of Herbrand's theorem

If $\exists \vec{x}.\, F(\vec{x})$ is true in all syntactic model, then there exists closed terms $\vec{t}_1, \ldots, \vec{t}_k$ such that $F(\vec{t}_1) \vee \ldots \vee F(\vec{t}_k)$ is provable.



conclude using the fan theorem

The interest of forcing

g is here a generic model *i.e.* a generic branch

$\rightsquigarrow$ no need to give *a priori* an order on atoms

$\rightsquigarrow$ forcing takes care of the tree structure

The interest of forcing

g is here a generic model *i.e.* a generic branch

$\rightsquigarrow$ no need to give *a priori* an order on atoms
$\rightsquigarrow$ forcing takes care of the tree structure

Our forcing structure:

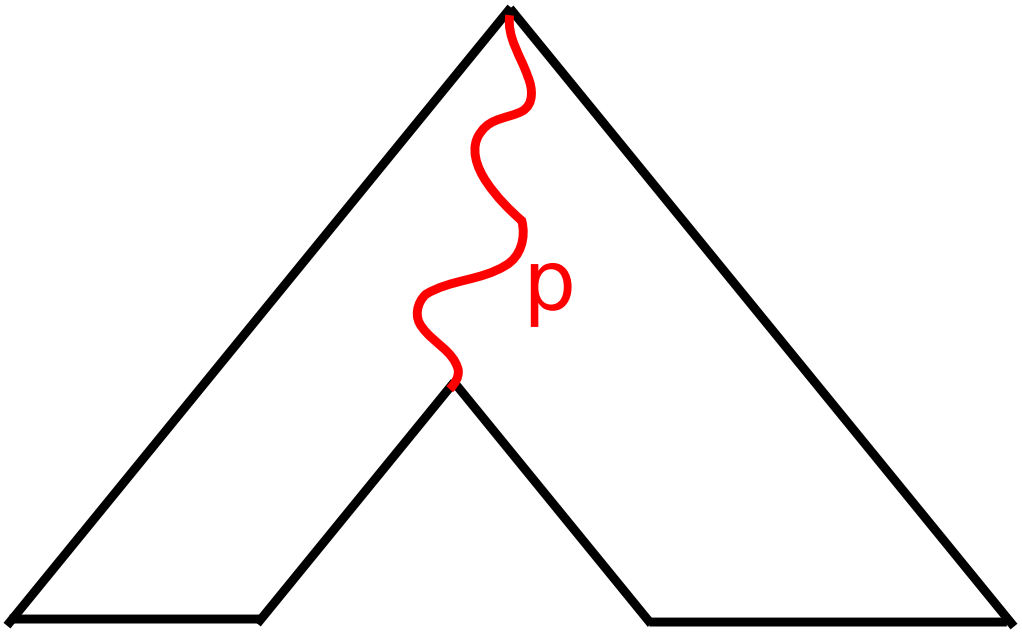
$$\begin{aligned}
\text{forcing conditions} &:= \text{finite functions from Atom to Bool} \\
\kappa &:= \iota \\
C[p] &:= p \in \mathrm{FVal} \wedge k \\
p \cdot q &:= p \cup q \\
1 &:= \varnothing \\
G &= \text{pairwise compatible conditions}
\end{aligned}$$

The computational content of forcing conditions

$$p \Vdash A := \forall r.\, C[p \cdot r] \Rightarrow A^*\, r$$
$$C[p] := p \in \mathsf{FVal} \wedge k$$

The computational content of forcing conditions

$$p \Vdash A := \forall r.\, C[p \cdot r] \Rightarrow A^*\, r$$
$$C[p] := p \in \mathsf{FVal} \wedge k$$



The totality axiom

Used instead of genericity

$p \Vdash \forall a \in \mathrm{Atom}.\, \exists q \in G.\, \exists b \in \mathrm{Bool}.\ \mathrm{test}\, a\, b\, q = 1$

The totality axiom

Used instead of genericity

$p \Vdash \forall a \in \mathrm{Atom}.\, \exists q \in G.\, \exists b \in \mathrm{Bool}.\ \mathrm{test}\, a\, b\, q = 1$

2 cases:

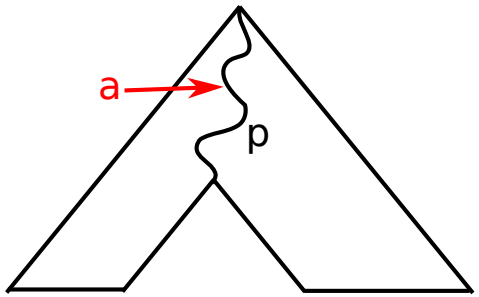
The totality axiom

Used instead of genericity

$p \Vdash \forall a \in \mathrm{Atom}.\, \exists q \in G.\, \exists b \in \mathrm{Bool}.\ \mathrm{test}\, a\, b\, q = 1$

2 cases:

- $a \in p$: answer b as in p



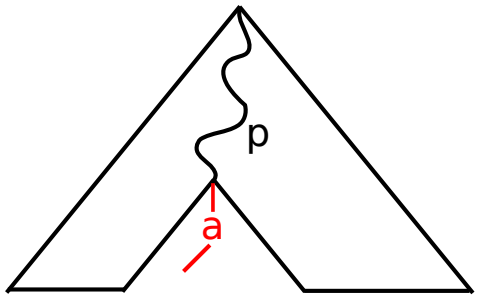
The totality axiom

Used instead of genericity

$p \Vdash \forall a \in \mathrm{Atom}.\, \exists q \in G.\, \exists b \in \mathrm{Bool}.\ \mathrm{test}\, a\, b\, q = 1$

2 cases:

- $a \in p$: answer b as in p
- $a \notin p$: we try both true and false



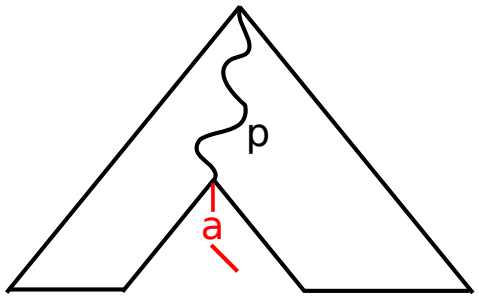
The totality axiom

Used instead of genericity

$p \Vdash \forall a \in \mathrm{Atom}.\, \exists q \in G.\, \exists b \in \mathrm{Bool}.\ \mathrm{test}\ a\ b\ q = 1$

2 cases:

- $a \in p$: answer b as in p
- $a \notin p$: we try both true and false



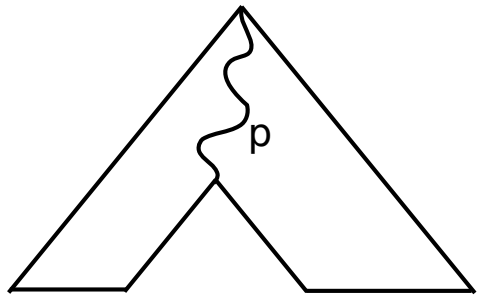
The totality axiom

Used instead of genericity

$p \Vdash \forall a \in \mathrm{Atom}.\, \exists q \in G.\, \exists b \in \mathrm{Bool}.\ \mathrm{test}\, a\, b\, q = 1$

2 cases:

- $a \in p$: answer b as in p
- $a \notin p$: we try both true and false



The program realizing the totality axiom

$$
\begin{array}{l}
\lambda caf.\ \mathrm{let}\ p, t := \alpha\, c\ \mathrm{in} \\
\quad \mathrm{if}\ \mathrm{Tot}_{\mathrm{test}}\ a'\ \mathrm{true}\ p\ \mathrm{then}\ f\,(\alpha\, c)\ \mathsf{I}\ \mathrm{true}^*\ \mathsf{I}^*\ \mathrm{else} \\
\quad \mathrm{if}\ \mathrm{Tot}_{\mathrm{test}}\ a'\ \mathrm{false}\ p\ \mathrm{then}\ f\,(\alpha\, c)\ \mathsf{I}\ \mathrm{false}^*\ \mathsf{I}^*\ \mathrm{else} \\
\quad f\,\langle \mathrm{Up}_{\mathrm{FVal}}\,((a')^+ \cup p), \lambda u. \\
\qquad f\,\langle \mathrm{Up}_{\mathrm{FVal}}\,((a')^- \cup p), \lambda v. \\
\qquad\quad t\,(\mathrm{merge}\, a'\, u\, v)\rangle\ \mathsf{I}\ \mathrm{false}^*\ \mathsf{I}^*\rangle\ \mathsf{I}\ \mathrm{true}^*\ \mathsf{I}^*
\end{array}
$$

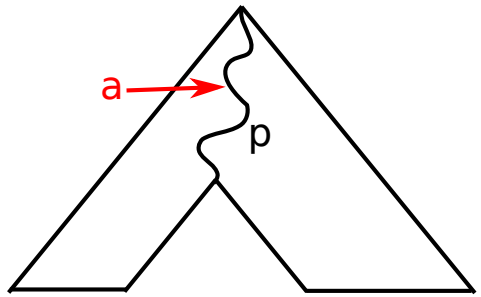
The totality axiom

Used instead of genericity

$p \Vdash \forall a \in \mathrm{Atom}.\, \exists q \in G.\, \exists b \in \mathrm{Bool}.\ \mathrm{test}\, a\, b\, q = 1$

2 cases:

- $a \in p$: answer b as in p
- $a \notin p$: we try both true and false



The program realizing the totality axiom

$$\begin{aligned}
&\lambda caf.\ \mathrm{let}\ p, t := \alpha\, c\ \mathrm{in}\\
&\quad \mathrm{if}\ \mathrm{Tot}_{\mathrm{test}}\ a'\ \mathrm{true}\ p\ \mathrm{then}\ f\,(\alpha\, c)\ \mathrm{I}\ \mathrm{true}^*\ \mathrm{I}^*\ \mathrm{else}\\
&\quad \mathrm{if}\ \mathrm{Tot}_{\mathrm{test}}\ a'\ \mathrm{false}\ p\ \mathrm{then}\ f\,(\alpha\, c)\ \mathrm{I}\ \mathrm{false}^*\ \mathrm{I}^*\ \mathrm{else}\\
&\quad f\,\langle \mathrm{Up}_{\mathrm{FVal}}\,((a')^+ \cup p), \lambda u.\\
&\qquad f\,\langle \mathrm{Up}_{\mathrm{FVal}}\,((a')^- \cup p), \lambda v.\\
&\qquad\quad t\,(\mathrm{merge}\, a'\, u\, v)\rangle\ \mathrm{I}\ \mathrm{false}^*\ \mathrm{I}^*\rangle\ \mathrm{I}\ \mathrm{true}^*\ \mathrm{I}^*
\end{aligned}$$

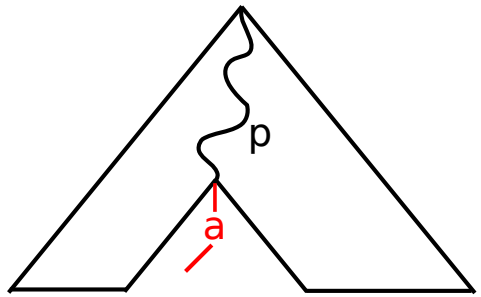
The totality axiom

Used instead of genericity

$p \Vdash \forall a \in \mathrm{Atom}.\, \exists q \in G.\, \exists b \in \mathrm{Bool}.\ \mathrm{test}\, a\, b\, q = 1$

2 cases:

- $a \in p$: answer b as in p
- $a \notin p$: we try both true and false



The program realizing the totality axiom

$$
\begin{array}{l}
\lambda caf.\ \mathrm{let}\ p, t := \alpha\, c\ \mathrm{in} \\
\quad \mathrm{if}\ \ \mathrm{Tot}_{\mathrm{test}}\ a'\ \mathrm{true}\ p\ \mathrm{then}\ f\,(\alpha\, c)\ \mathrm{I}\ \mathrm{true}^*\ \mathrm{I}^*\ \ \mathrm{else} \\
\quad \mathrm{if}\ \ \mathrm{Tot}_{\mathrm{test}}\ a'\ \mathrm{false}\ p\ \mathrm{then}\ f\,(\alpha\, c)\ \mathrm{I}\ \mathrm{false}^*\ \mathrm{I}^*\ \ \mathrm{else} \\
\quad f\,\langle \mathrm{Up}_{\mathrm{FVal}}\,((a')^{+} \cup p), \lambda u. \\
\qquad f\,\langle \mathrm{Up}_{\mathrm{FVal}}\,((a')^{-} \cup p), \lambda v. \\
\qquad\quad t\,(\mathrm{merge}\, a'\, u\, v)\rangle\ \mathrm{I}\ \mathrm{false}^*\ \mathrm{I}^*\rangle\ \mathrm{I}\ \mathrm{true}^*\ \mathrm{I}^*
\end{array}
$$

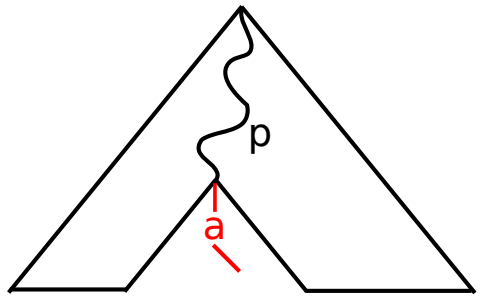
The totality axiom

Used instead of genericity

$p \Vdash \forall a \in \mathrm{Atom}.\, \exists q \in G.\, \exists b \in \mathrm{Bool}.\ \mathrm{test}\, a\, b\, q = 1$

2 cases:

- $a \in p$: answer b as in p
- $a \notin p$: we try both true and false



The program realizing the totality axiom

$$
\begin{array}{l}
\lambda caf.\ \mathrm{let}\ p, t := \alpha\, c\ \mathrm{in} \\
\quad \mathrm{if}\ \mathrm{Tot}_{\mathrm{test}}\ a'\ \mathrm{true}\ p\ \mathrm{then}\ f\,(\alpha\, c)\ \mathrm{I}\ \mathrm{true}^*\ \mathrm{I}^*\ \mathrm{else} \\
\quad \mathrm{if}\ \mathrm{Tot}_{\mathrm{test}}\ a'\ \mathrm{false}\ p\ \mathrm{then}\ f\,(\alpha\, c)\ \mathrm{I}\ \mathrm{false}^*\ \mathrm{I}^*\ \mathrm{else} \\
\quad f\,\langle \mathrm{Up}_{\mathrm{FVal}}\,((a')^+ \cup p), \lambda u. \\
\qquad f\,\langle \mathrm{Up}_{\mathrm{FVal}}\,((a')^- \cup p), \lambda v. \\
\qquad\quad t\,(\mathrm{merge}\, a'\, u\, v)\rangle\ \mathrm{I}\ \mathrm{false}^*\ \mathrm{I}^*\rangle\ \mathrm{I}\ \mathrm{true}^*\ \mathrm{I}^*
\end{array}
$$

$\mathrm{PA}\omega_v^+$

Sorts

$$\tau, \sigma \quad := \quad \iota \mid o \mid \tau \to \sigma$$

Values

$$v^\iota \quad := \quad \underline{x} \mid C(\vec{v})$$

Math. expr. (types)

$$\begin{aligned} M, N, A, B \quad := \quad & x^\tau \mid \lambda x^\tau.\, M \mid M\,N \\ & \mid 0 \mid S \mid \mathrm{rec}_\tau \\ & \mid A \Rightarrow B \mid \forall x^\tau.\, A \mid M \doteq_\tau N \mapsto A \\ & \mid V \Rightarrow_v B \mid \forall \underline{x}^\iota.\, A \end{aligned}$$

Proof-terms

$$t, u \quad := \quad x \mid \lambda x.\, t \mid t\,u \mid \mathrm{callcc}$$

Contexts

$$\Gamma \quad := \quad \varnothing \mid \Gamma, x : A \mid \Gamma, x : v \in V$$

Judgments

$$\mathcal{E}; \Gamma \vdash t : A$$
$$\mathcal{E}; \Gamma \vdash v \in V$$

Values in the KAM

Can only be on the stack (no evaluation rule)
$\Rightarrow$ no need for falsity value, only truth value
$\Rightarrow$ when we want to return a value, we use CPS:

$$\overline{V} := \forall Z^o.\,(V \Rightarrow_v Z) \Rightarrow Z$$

New data judgment: $\mathcal{E}; \Gamma \vdash v \in V$
$\Rightarrow$ intuition: v directly belong to V,
not reduces to a term in V

each data can allow several representation (as proof terms)
ex: sets *vs.* lists

Forcing with values

2 possibilities:

- values completely transparent for forcing
  $\Rightarrow$ 2 translations for $\lambda x.\, t$ / $t\ u$ if x / u value or not
  $\Rightarrow$ no forcing condition attached to values in KFAM
  $\Rightarrow$ 2 λ, 2 @, (2 cons on stack?)
  $\Rightarrow$ real separation between data and closures
- values have a (dummy) forcing condition
  $\Rightarrow$ only 1 α for λ / @
  $\Rightarrow$ still 2 λ

In both cases, we have $p \Vdash V \Rightarrow_v A \iff V \Rightarrow_v p \Vdash A$

Forcing term translation with values

$$(x^\tau)^* := x^{\tau^*} \qquad 0^* := 0$$
$$\lambda x^\tau.\, M := \lambda x^{\tau^*}.\, M^* \qquad S^* := S$$
$$(M\,N)^* := M^*\,N^* \qquad \mathrm{rec}^*_\tau := \mathrm{rec}_{\tau^*}$$

$$(\forall x^\tau.\, A)^* := \lambda r^\iota.\, \forall x^{\tau^*}.\, A^*\, r$$
$$(M \doteq N \mapsto A)^* := \lambda r^\iota.\, M^* \doteq N^* \mapsto A^*\, r$$
$$(V \Rightarrow_v A)^* := \lambda r^\iota.\, V \Rightarrow_v A^*\, r$$
$$(A \Rightarrow B)^* := \lambda r^\iota.\, \forall q^\iota \forall (r')^\iota.\, r \doteq qr' \mapsto (\forall s^\iota.\, C[qs] \Rightarrow A^*\, s) \Rightarrow B^*\, r'$$

Forcing term translation with values

$$(x^\tau)^* := x^{\tau^*} \qquad 0^* := 0$$
$$\lambda x^\tau.\, M := \lambda x^{\tau^*}.\, M^* \qquad S^* := S$$
$$(M\,N)^* := M^*\,N^* \qquad \mathrm{rec}_\tau^* := \mathrm{rec}_{\tau^*}$$

$$(\forall x^\tau.\, A)^* := \lambda r^\iota.\, \forall x^{\tau^*}.\, A^*\, r$$
$$(M \doteq N \mapsto A)^* := \lambda r^\iota.\, M^* \doteq N^* \mapsto A^*\, r$$
$$(V \Rightarrow_v A)^* := \lambda r^\iota.\, \forall q^\iota \forall (r')^\iota.\, r \doteq qr' \mapsto V \Rightarrow_v A^*\, r$$
$$(A \Rightarrow B)^* := \lambda r^\iota.\, \forall q^\iota \forall (r')^\iota.\, r \doteq qr' \mapsto (\forall s^\iota.\, C[qs] \Rightarrow A^*\, s) \Rightarrow B^*\, r'$$