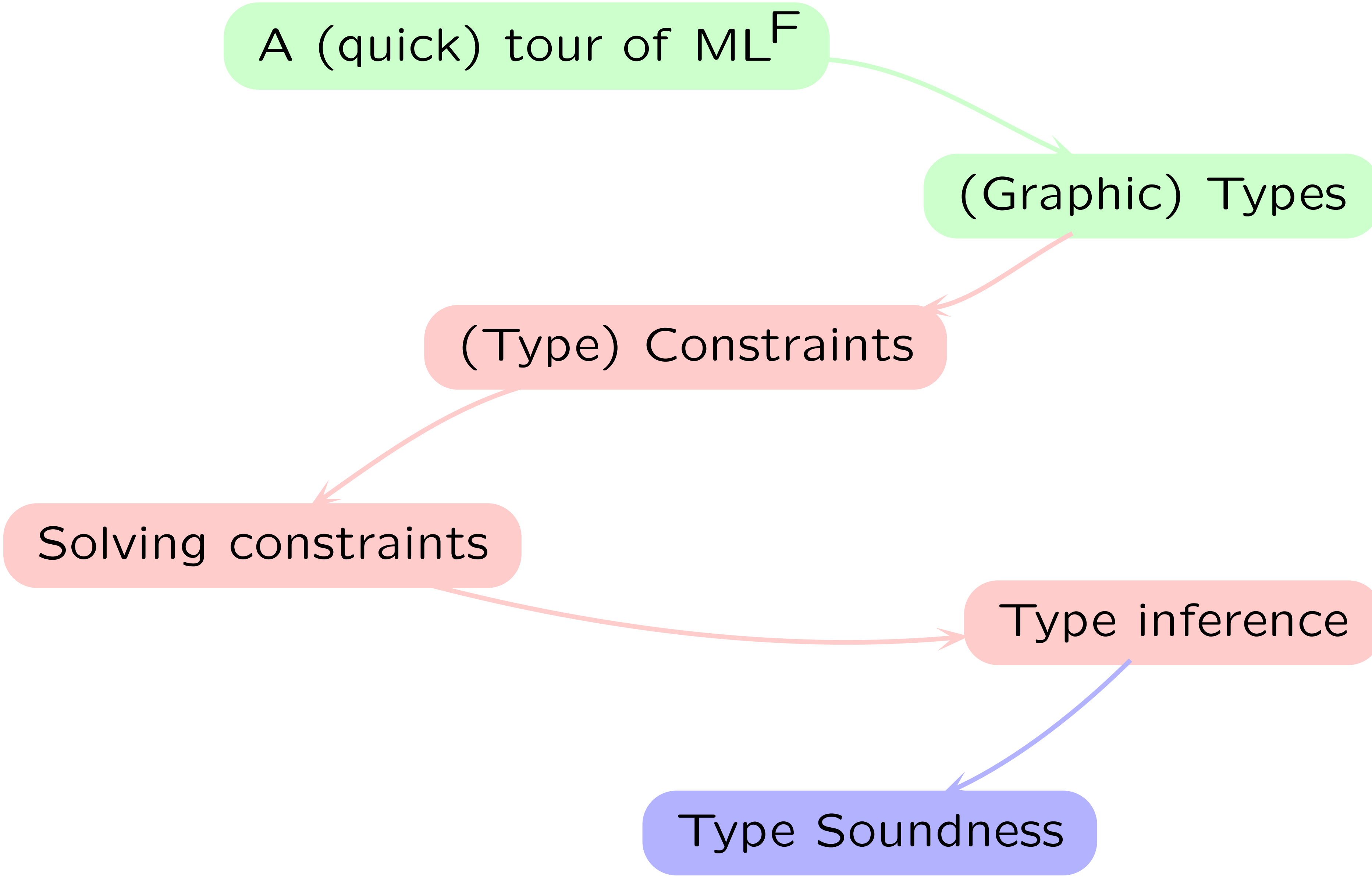

A (quick) tour of ML^F
(Graphic) Types
(Type) Constraints
Solving constraints
Type inference
Type Soundness

A Fully Graphical Presentation of ML^{F}

Didier Rémy

&

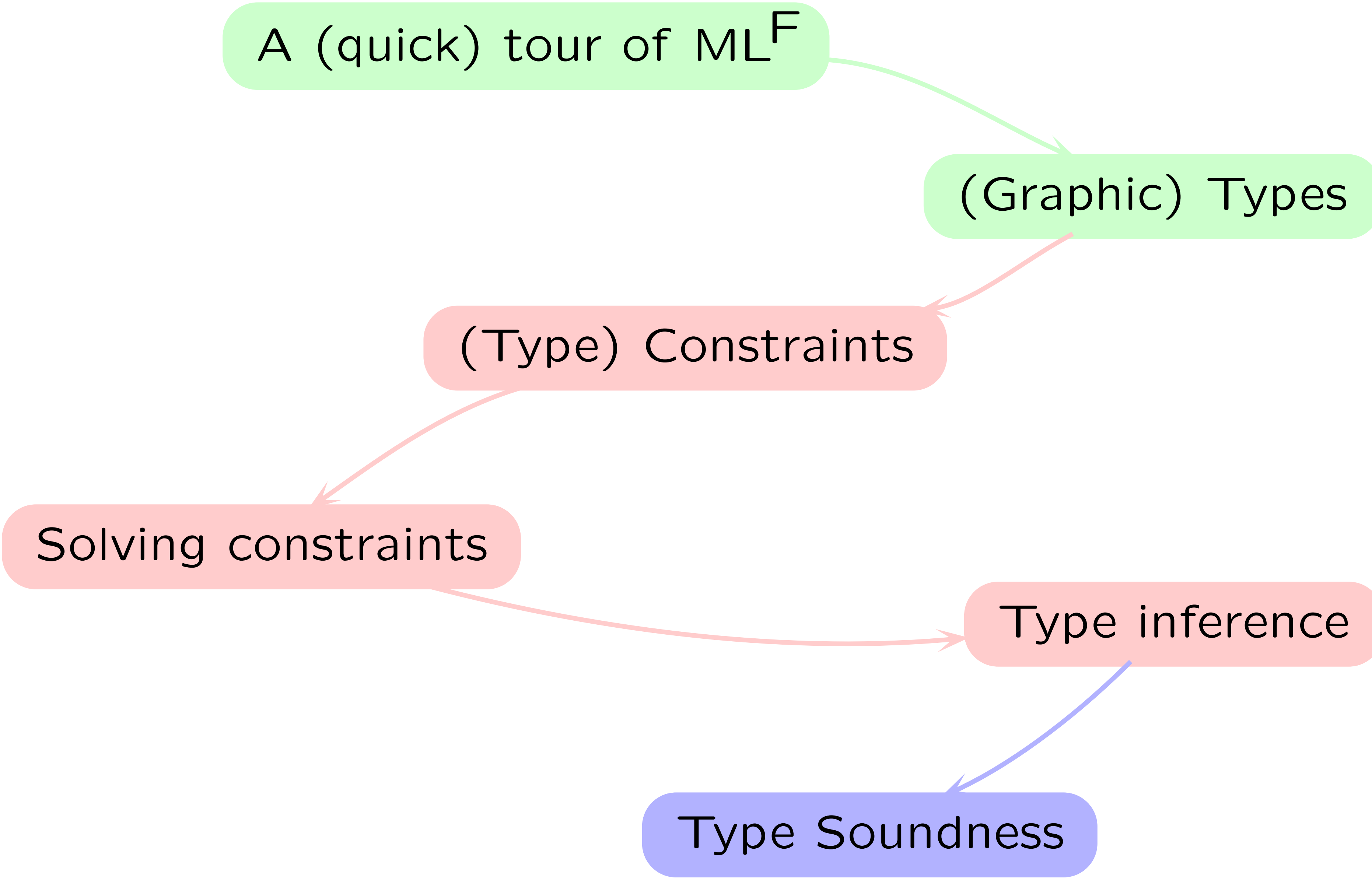
Boris Yakobowski

Didier

Le Botlan

INRIA

Rocquencourt


A (quick) tour of ML^F
(Graphic) Types
(Type) Constraints
Solving constraints
Type inference
Type Soundness

Motivations

The original ML^{F}

- It is *intuitively* simple, but
- Its *purely syntactic* presentation is *technically* involved.

Is it the right definition?

(It is sound, indeed, but where there other better choices?)

No, it is not quite right!

We now have the right definition, twice:

- **"Semantically", in Recasting MLF (work with Didier Le Botlan)**
- **Graphically, in this work**

This work

A new fully graphical presentation of Full ML^{F}

We build on previous work

A Graphical presentation of ML^{F} types (TLDI 06)

Types and the instance relation are either well-known or simple operations on graphs. Allows for an efficient unification algorithm.

We

- enrich types with type constraints.
- solve type constraints by reducing them to unification problems.
- express type inference as typing constraints
- obtain a type-inference algorithm about as efficient as the one for ML
- show type soundness by reasoning on graphical typing constraints.

The key to ML$^{\mathsf{F}}$

let choose $= \lambda(x)\ \lambda(y)$ **if** $true$ **then** x **else** $y : \forall\, \alpha \cdot \alpha \to \alpha \to \alpha$

let $id = \lambda(z)\ z : \forall\, \alpha \cdot \alpha \to \alpha$

choose $(\lambda(x)\ x)$:

The key to ML$^{\mathsf{F}}$

let choose $= \lambda(x)\ \lambda(y)$ **if** $true$ **then** x **else** $y : \forall\, \alpha \cdot \alpha \to \alpha \to \alpha$

let $id = \lambda(z)\ z : \forall\, \alpha \cdot \alpha \to \alpha$

choose $(\lambda(x)\ x)$: $\left\{ \begin{array}{l} \forall\, \alpha \cdot (\alpha \to \alpha) \to (\alpha \to \alpha) \\ (\forall\, \alpha \cdot \alpha \to \alpha) \to (\forall\, \alpha \cdot \alpha \to \alpha) \end{array} \right.$

The key to ML^{F}

let choose $= \lambda(x)\ \lambda(y)$ **if** $true$ **then** x **else** $y : \forall\, \alpha \cdot \alpha \to \alpha \to \alpha$

let $id = \lambda(z)\ z : \forall\, \alpha \cdot \alpha \to \alpha$

choose $(\lambda(x)\ x)$: $\left\{ \begin{array}{l} \forall\, \alpha \cdot (\alpha \to \alpha) \to (\alpha \to \alpha) \\ (\forall\, \alpha \cdot \alpha \to \alpha) \to (\forall\, \alpha \cdot \alpha \to \alpha) \end{array} \right\}$ No better choice in F

The key to ML^{F}

let choose $= \lambda(x)\ \lambda(y)$ **if** $true$ **then** x **else** $y : \forall\,\alpha \cdot \alpha \to \alpha \to \alpha$

let $id = \lambda(z)\ z : \forall\,\alpha \cdot \alpha \to \alpha$

choose $(\lambda(x)\ x)$: $\left\{ \begin{array}{l} \forall\,\alpha \cdot (\alpha \to \alpha) \to (\alpha \to \alpha) \\ (\forall\,\alpha \cdot \alpha \to \alpha) \to (\forall\,\alpha \cdot \alpha \to \alpha) \end{array} \right\}$ No better choice in F

$$: \quad \forall(\beta > \forall(\alpha)\,\alpha \to \alpha)\,\beta \to \beta \quad \text{in } \mathsf{ML}^{\mathsf{F}}$$

The key to ML^{F}

let choose $= \lambda(x)\ \lambda(y)$ **if** $true$ **then** x **else** $y : \forall\,\alpha \cdot \alpha \to \alpha \to \alpha$

let $id = \lambda(z)\ z : \forall\,\alpha \cdot \alpha \to \alpha$

choose $(\lambda(x)\ x)$ $: \left\{ \begin{array}{l} \forall\,\alpha \cdot (\alpha \to \alpha) \to (\alpha \to \alpha) \\ (\forall\,\alpha \cdot \alpha \to \alpha) \to (\forall\,\alpha \cdot \alpha \to \alpha) \end{array} \right\}$ No better choice in F

$$: \quad \forall(\beta > \forall(\alpha)\,\alpha \to \alpha)\,\beta \to \beta \quad \text{in } \mathsf{ML}^{\mathsf{F}}$$

$$\leqslant \left\{ \begin{array}{l} \forall(\beta = \forall(\alpha)\,\alpha \to \alpha)\,\beta \to \beta \\ \forall(\alpha)\,\forall(\beta = \alpha \to \alpha)\,\beta \to \beta \end{array} \right.$$

The key to ML^{F}

let choose $= \lambda(x)\ \lambda(y)$ **if** $true$ **then** x **else** $y : \forall\,\alpha \cdot \alpha \to \alpha \to \alpha$

let $id = \lambda(z)\ z : \forall\,\alpha \cdot \alpha \to \alpha$

choose $(\lambda(x)\ x)\ : \left\{ \begin{array}{l} \forall\,\alpha \cdot (\alpha \to \alpha) \to (\alpha \to \alpha) \\ (\forall\,\alpha \cdot \alpha \to \alpha) \to (\forall\,\alpha \cdot \alpha \to \alpha) \end{array} \right\}$ No better choice in F

$$: \quad \forall(\beta > \forall(\alpha)\,\alpha \to \alpha)\,\beta \to \beta \quad \text{in } \mathsf{ML}^{\mathsf{F}}$$

$$\leqslant \left\{ \begin{array}{l} \forall(\beta = \forall(\alpha)\,\alpha \to \alpha)\,\beta \to \beta \\ \forall(\alpha)\,\forall(\beta = \alpha \to \alpha)\,\beta \to \beta \end{array} \right.$$

But

$\lambda(x)\ x\ x$	:	ill-typed, as we do not guess polymorphism!
$\lambda(x : \forall(\alpha)\,\alpha \to \alpha)\ x\ x$	:	$\forall(\beta = \forall(\alpha)\,\alpha \to \alpha)\,\beta \to \beta$

The key to ML^F

let choose $= \lambda(x)\ \lambda(y)$ **if** $true$ **then** x **else** $y : \forall\,\alpha\cdot\alpha\to\alpha\to\alpha$

let $id = \lambda(z)\ z : \forall\,\alpha\cdot\alpha\to\alpha$

choose $(\lambda(x)\ x)$: $\left\{\begin{array}{l}\forall\,\alpha\cdot(\alpha\to\alpha)\to(\alpha\to\alpha)\\(\forall\,\alpha\cdot\alpha\to\alpha)\to(\forall\,\alpha\cdot\alpha\to\alpha)\end{array}\right\}$ No better choice in F

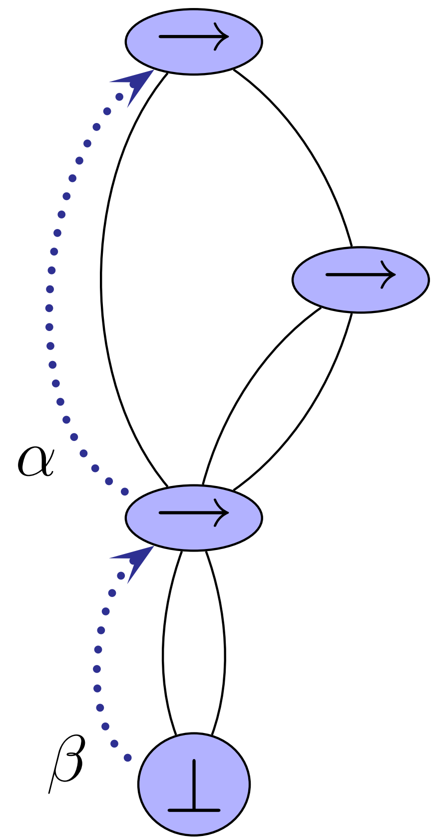
Function parameters that are used polymorphically and only those need an annotation.

But

$\lambda(x)\ x\ x$: ill-typed, as we do not guess polymorphism!

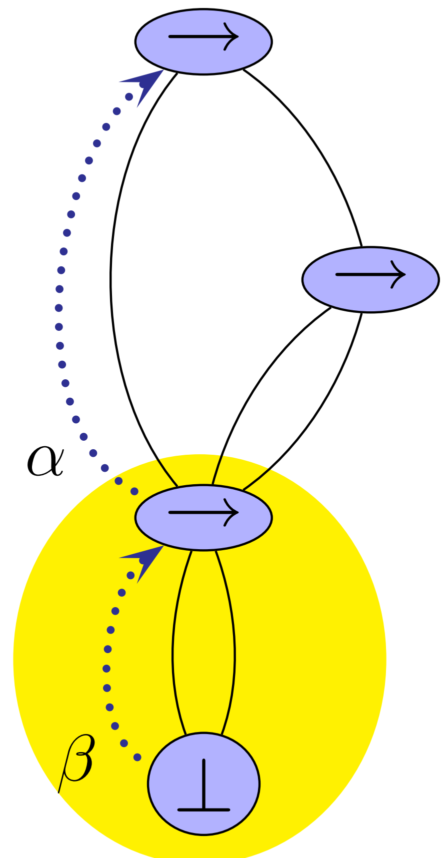
$\lambda(x:\forall(\alpha)\,\alpha\to\alpha)\ x\ x$: $\forall(\beta=\forall(\alpha)\,\alpha\to\alpha)\,\beta\to\beta$

Graphical Types



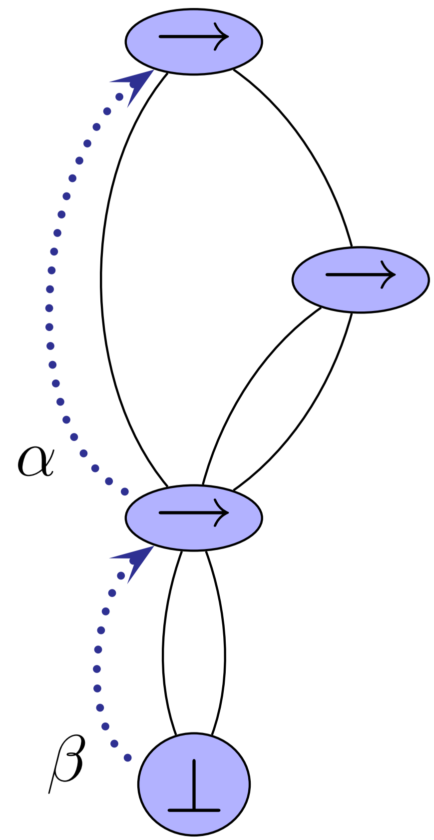
$$\forall\Big(\alpha \geq \forall(\beta)\,\beta \to \beta\Big)\,\alpha \to (\alpha \to \alpha)$$

Graphical Types



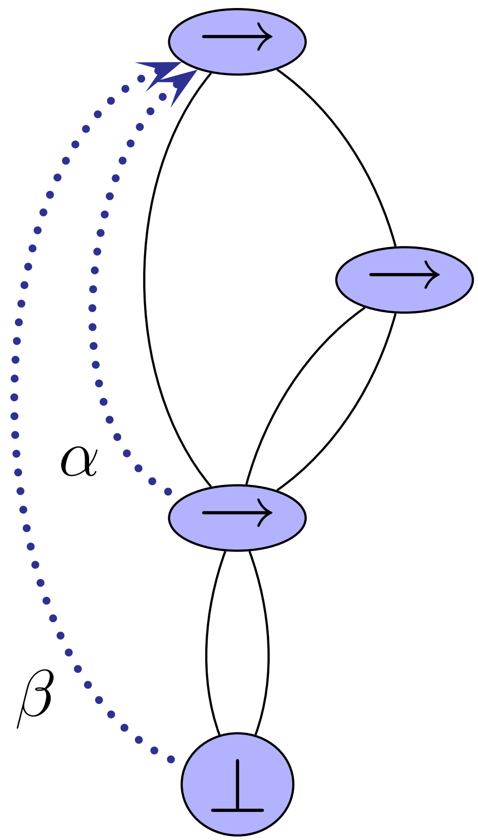
$$\forall\left(\alpha \geq \forall(\beta)\, \beta \to \beta\right)\, \alpha \to (\alpha \to \alpha)$$

Graphical Types



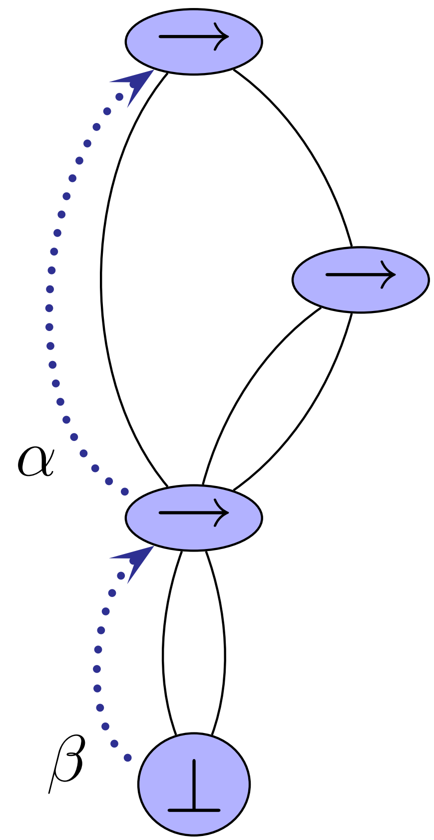
$$\forall\Big(\alpha \geq \forall(\beta)\,\beta \to \beta\Big)\,\alpha \to (\alpha \to \alpha)$$

Graphical Types



$$\forall(\beta)\,\forall\left(\alpha \geq \beta \to \beta\right)\,\alpha \to \alpha \to \alpha$$

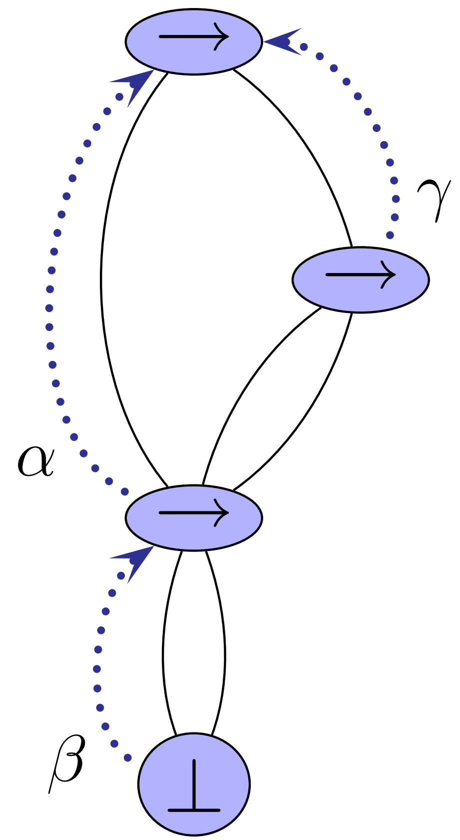
Graphical Types



$$\forall\left(\alpha \geq \forall(\beta)\,\beta \to \beta\right)\,\alpha \to (\alpha \to \alpha)$$

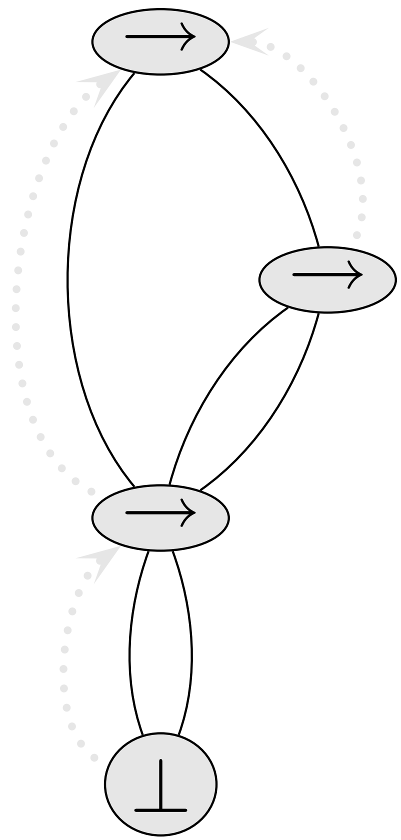
Graphical Types

Binding all nodes allows for a more regular representation



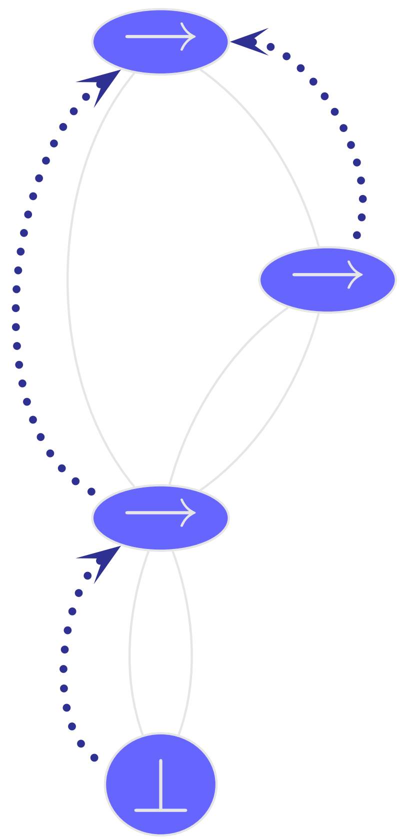
$$\forall\left(\alpha \geq \forall(\beta)\,\beta \to \beta\right)\forall\left(\gamma \geq \alpha \to \alpha\right)\alpha \to \gamma$$

Graphical Types



Superposition of a term-dag

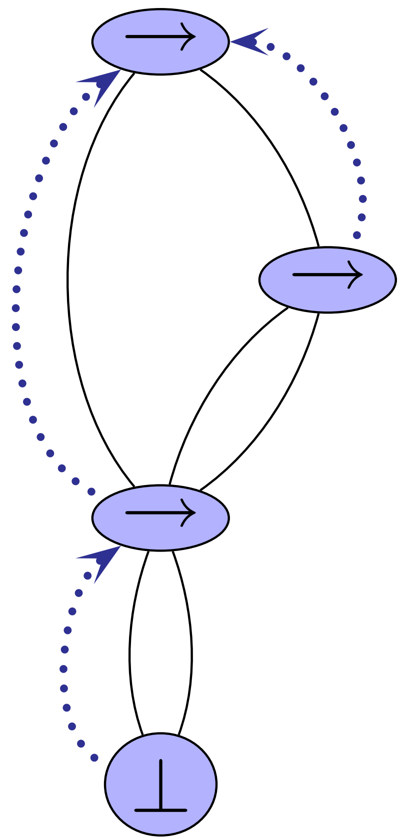
(first-order terms sharing suffixes)

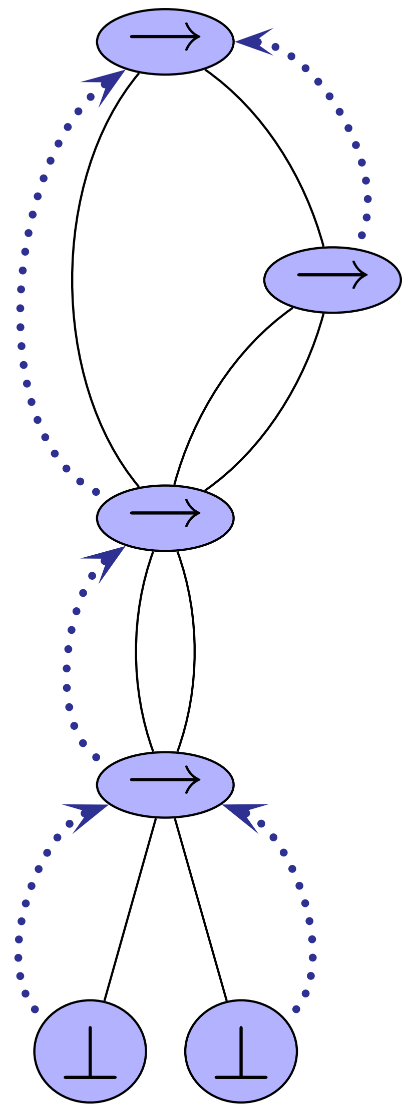


and a binding tree structure

(+ well-formedness conditions between both)

Graphical Types



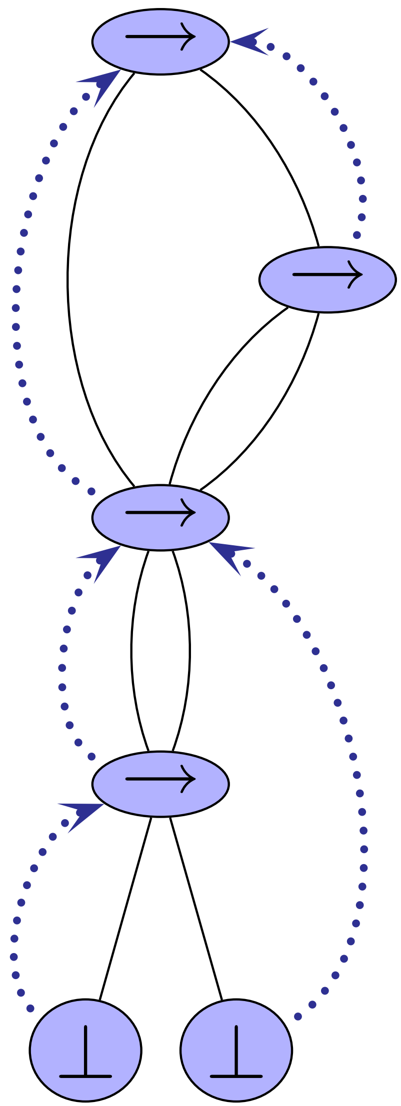


Instance relation

Grafting

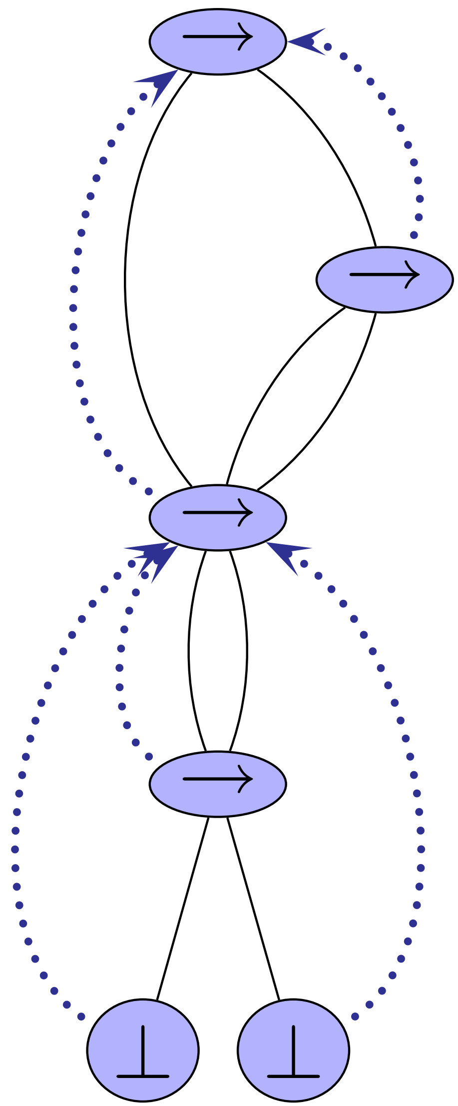
Instance relation

Raising



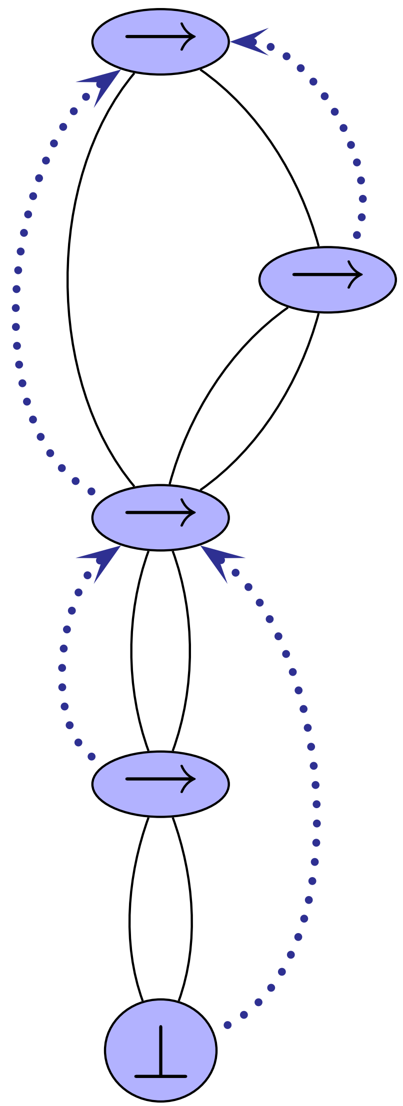
Instance relation

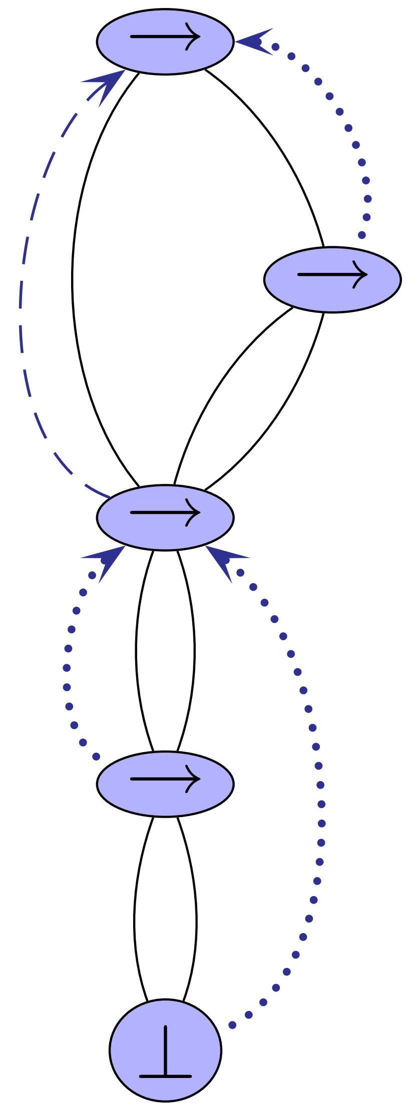
Raising



Instance relation

Merging





Instance relation

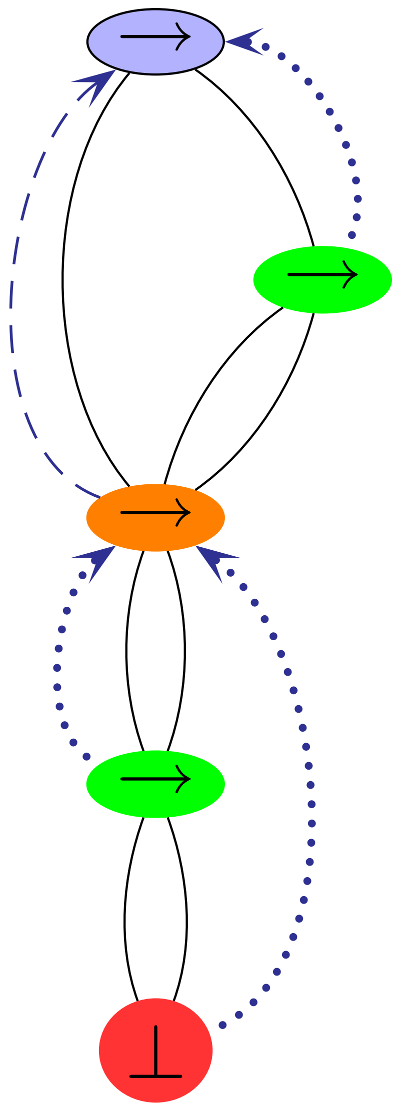
Weakening

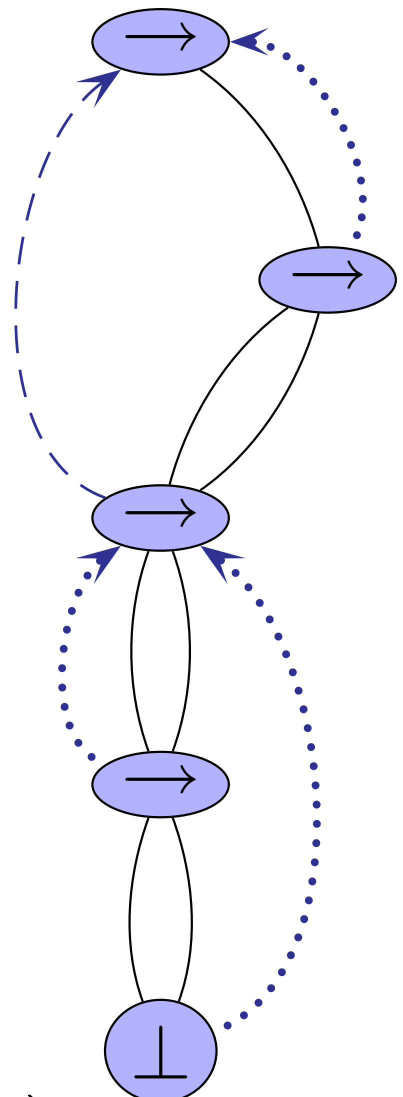
Changes permissions

Instance relation

Weakening

Changes permissions





Instance relation

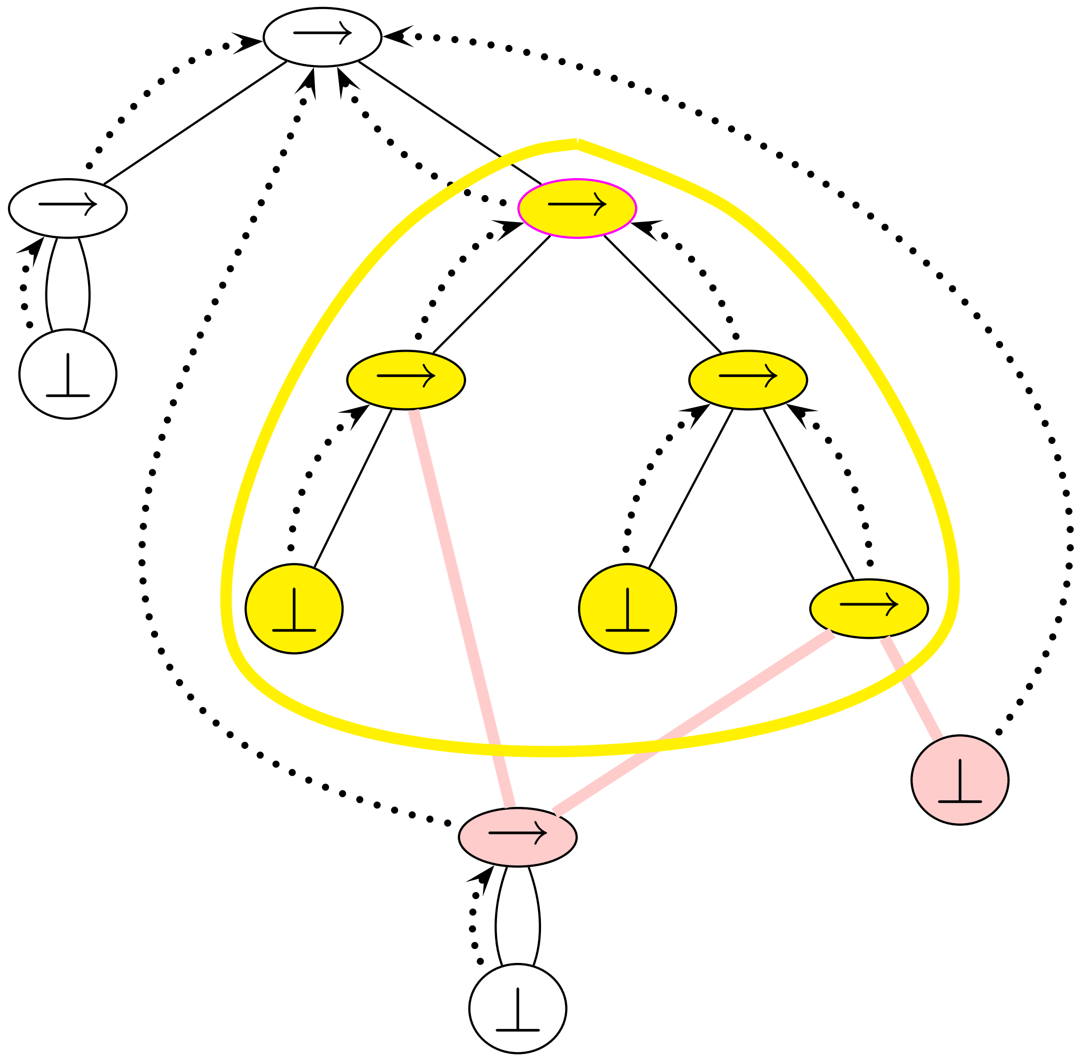
Weakening

Changes permissions

$$\forall\Big(\alpha = \forall(\beta)\,\beta \to \beta\Big)\;\forall\Big(\gamma \geq \alpha \to \alpha\Big)\;\alpha \to \gamma$$

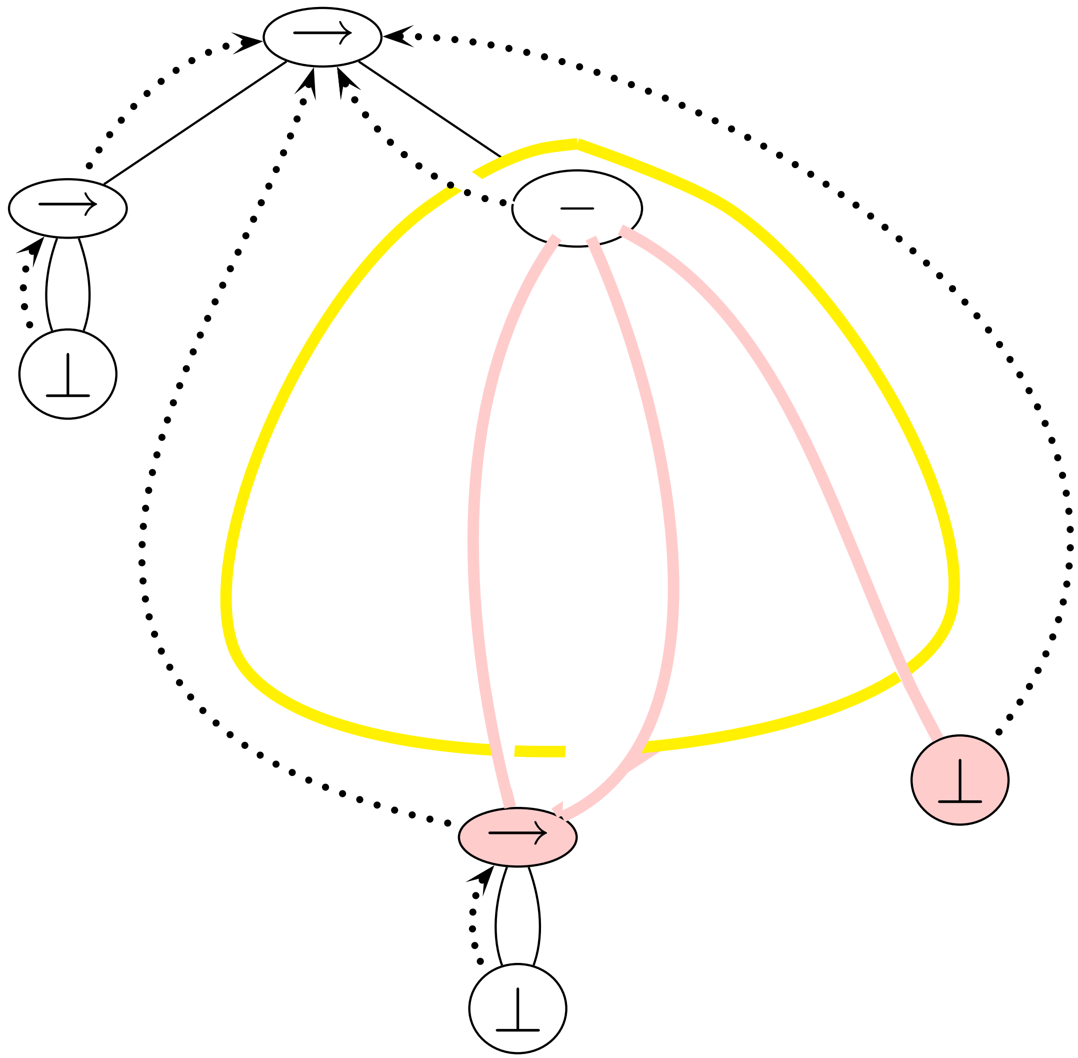
Inner nodes

The interior of a node n is the set of nodes, said inner n, that are transitively bound at n.



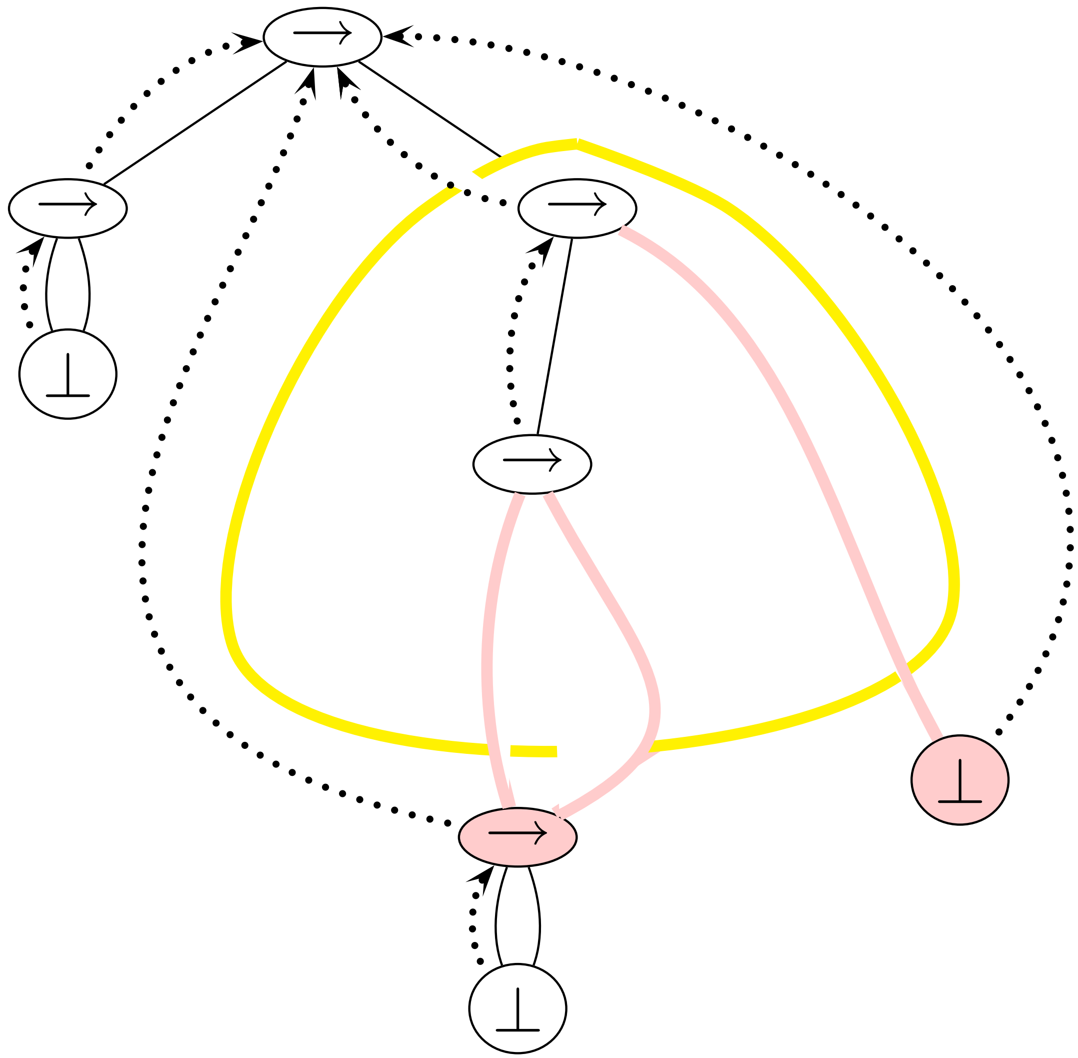
Inner nodes

It can be transformed *locally*, as long as the interface (struture edges crossing the frontier) is maintained.



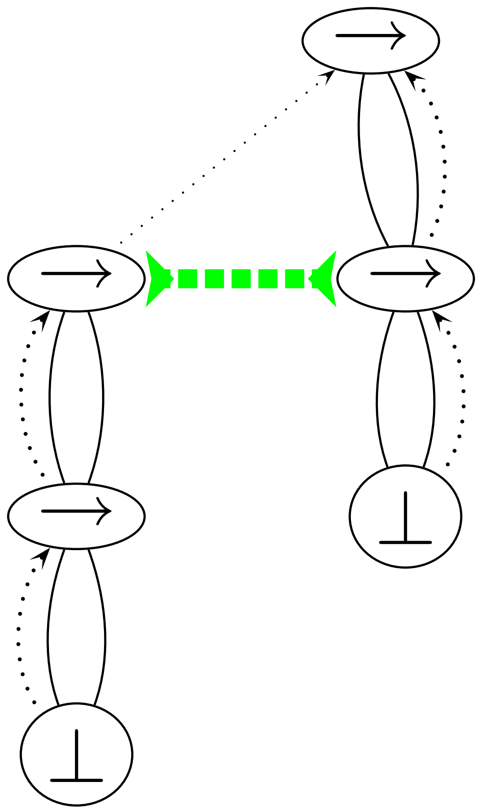
Inner nodes

It can be transformed *locally*, as long as the interface (struture edges crossing the frontier) is maintained.



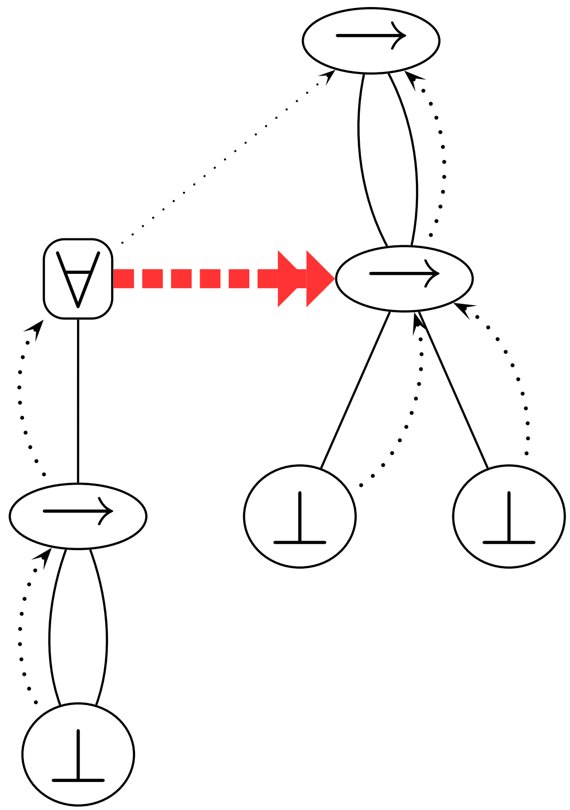
They are graphic types extended with. . .

1) Existential nodes and unification edges



They are graphic types extended with. . .

2) Type schemes and instantiation edges



Use sorts Scheme and Type to constraint formation of edges

General picture

- Constraints are given a meaning as a set of types that are solutions of the constraints.

  (Hence two constraints with the same meaning may have completely different shapes, e.g. *two unsolvable constraints are equivalent)*

- Constraints are also types. As such, they can be instantiated along $\sqsubseteq$.

Semantics of constraints

General picture

- Constraints are given a meaning as a set of types that are solutions of the constraints.

  (Hence two constraints with the same meaning may have completely different shapes, e.g. *two unsolvable constraints are equivalent)*

- Constraints are also types. As such, they can be instantiated along $\sqsubseteq$.

Definition:

- A presolution of a constraint χ is an instance of χ in which all constraint edge are **solved**.

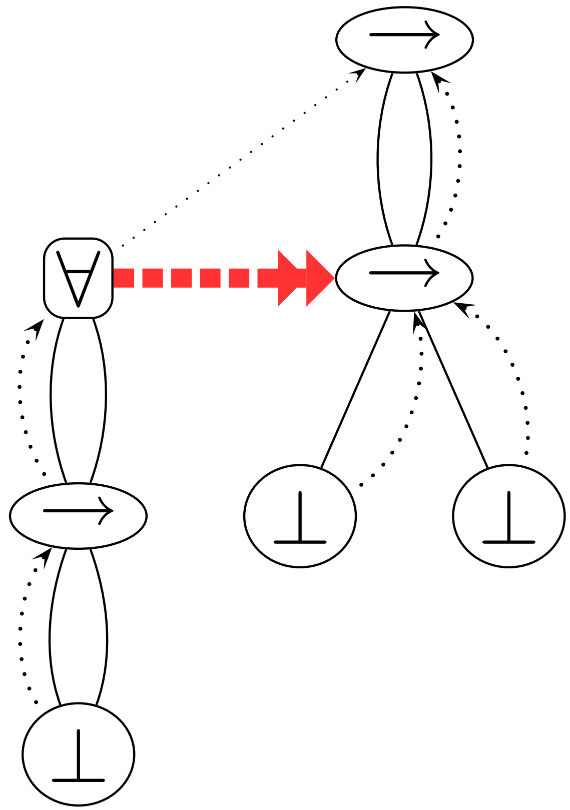
- A solution of χ is of a the **projection of** a presolution of χ

Projection of a constraint

1) Remove all existential nodes, 2) garbage collect from the root node, 3) remove all dangling edges.

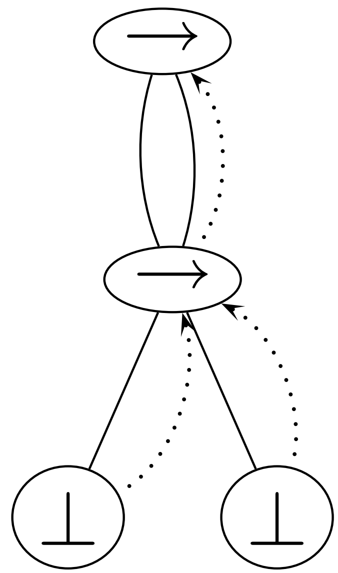
Projection of a constraint

1) Remove all existential nodes, 2) garbage collect from the root node, 3) remove all dangling edges.



Projection of a constraint

1) Remove all existential nodes, 2) garbage collect from the root node,
3) remove all dangling edges.



Semantics of constraints

A unification edge p q **if** $p = q$

Unification edges are solved by unification :-)

Semantics of constraints

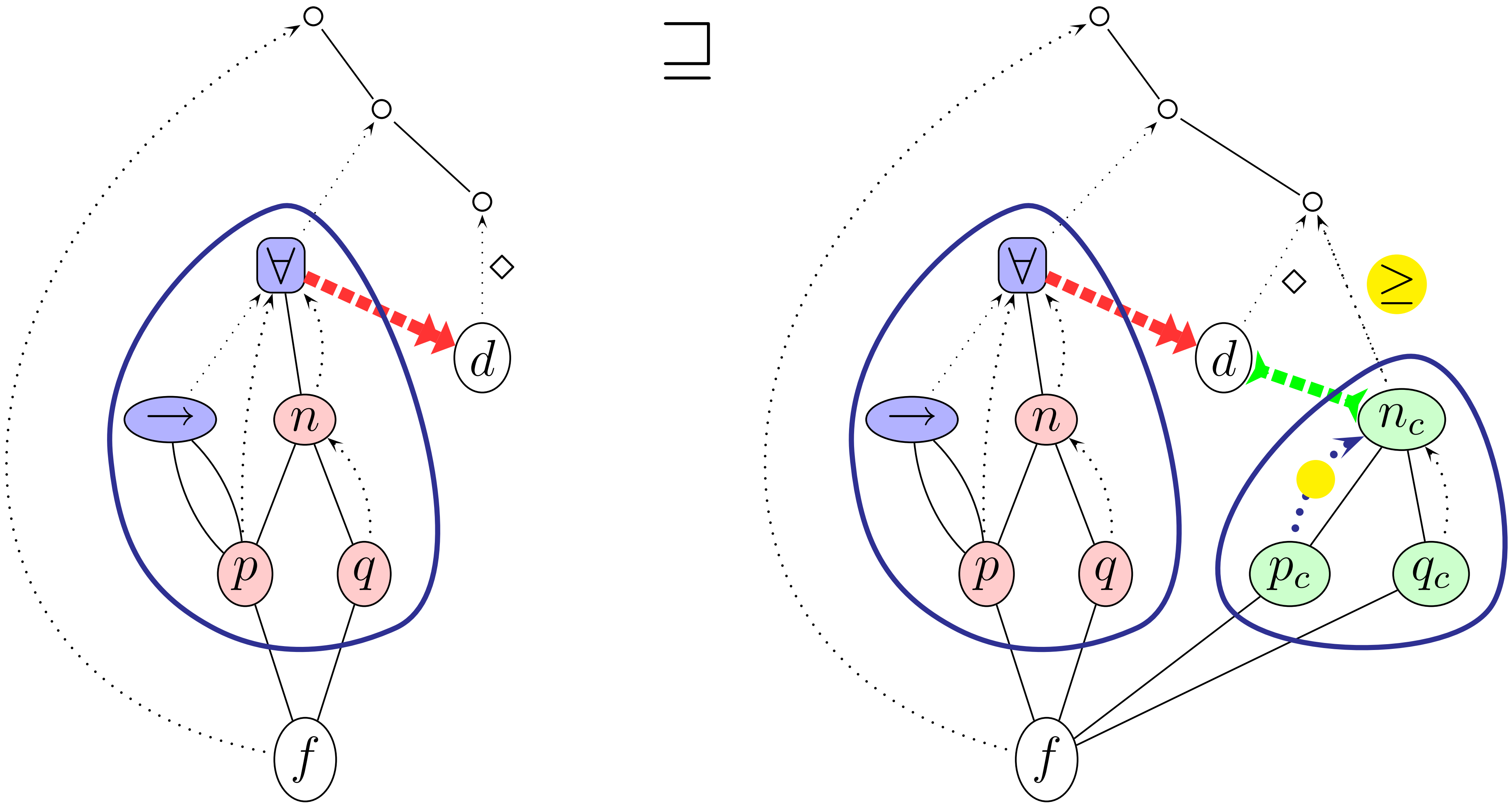
An instantiation edge $s \twoheadrightarrow m$ is solved if the type at m matches the type scheme at s

Semantics of constraints

An instantiation edge $s \dashrightarrow\!\!\!\!\twoheadrightarrow m$ is solved if
the type at m matches the type scheme at s

i.e. the unification of a copy of s with m leaves the constaint unchanged.

An instantiation edge s ⇢ m is solved if



A key property

A solvable constraint has a principal presolution,

i.e. an instance of the original constraint that is a presolution and of which all other presolutions are instances.

Entailment

$\chi \Vdash \chi'$ if all solutions of χ are solutions of χ'.

Entailment

$\chi \Vdash \chi'$ if all solutions of χ are solutions of χ'.

In particular $\chi \Vdash \chi'$ whenever $\chi' \sqsubseteq \chi$
(χ' is more constrained than χ.)

However, interesting entailments are not along $\sqsubseteq$.

$\chi \dashv\Vdash \chi'$ if the solutions of χ are exactly the solutions of χ'.

Sound and complete transformations

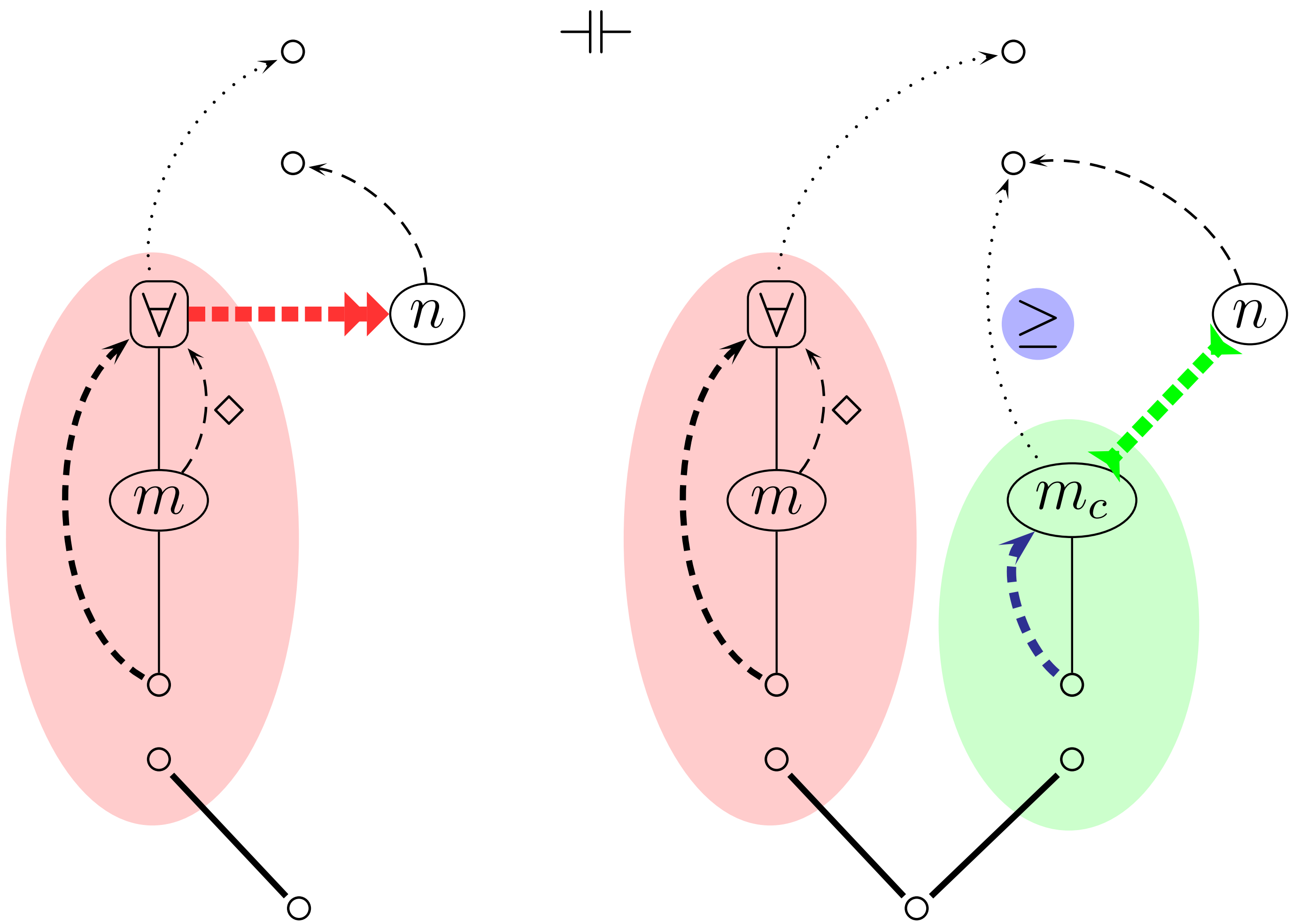
Unif-Elim

Solving unification edges

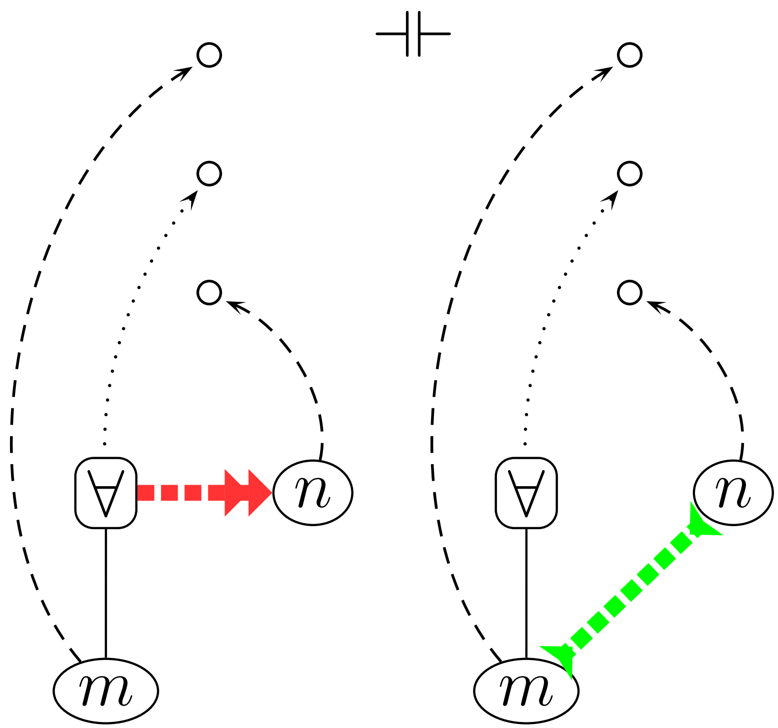
Exists-Elim

Elimination of existential nodes without inner constraints

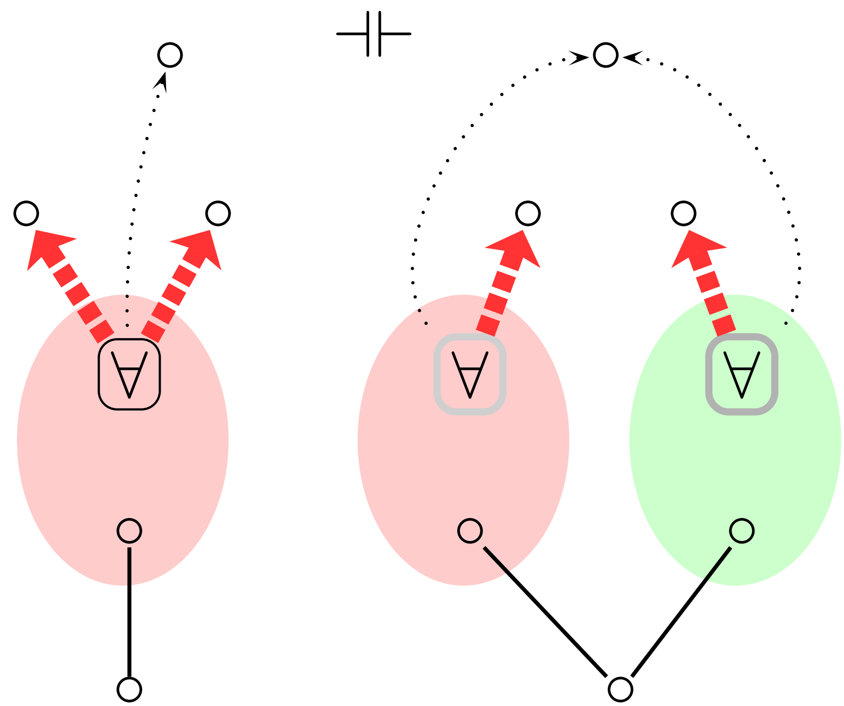
Inst-Elim-Poly (Existential nodes and constraint edges also copied)



Inst-Elim-Mono (Degenerate case



Inst-Copy (Existential nodes and constraint edges also copied)



A strategy

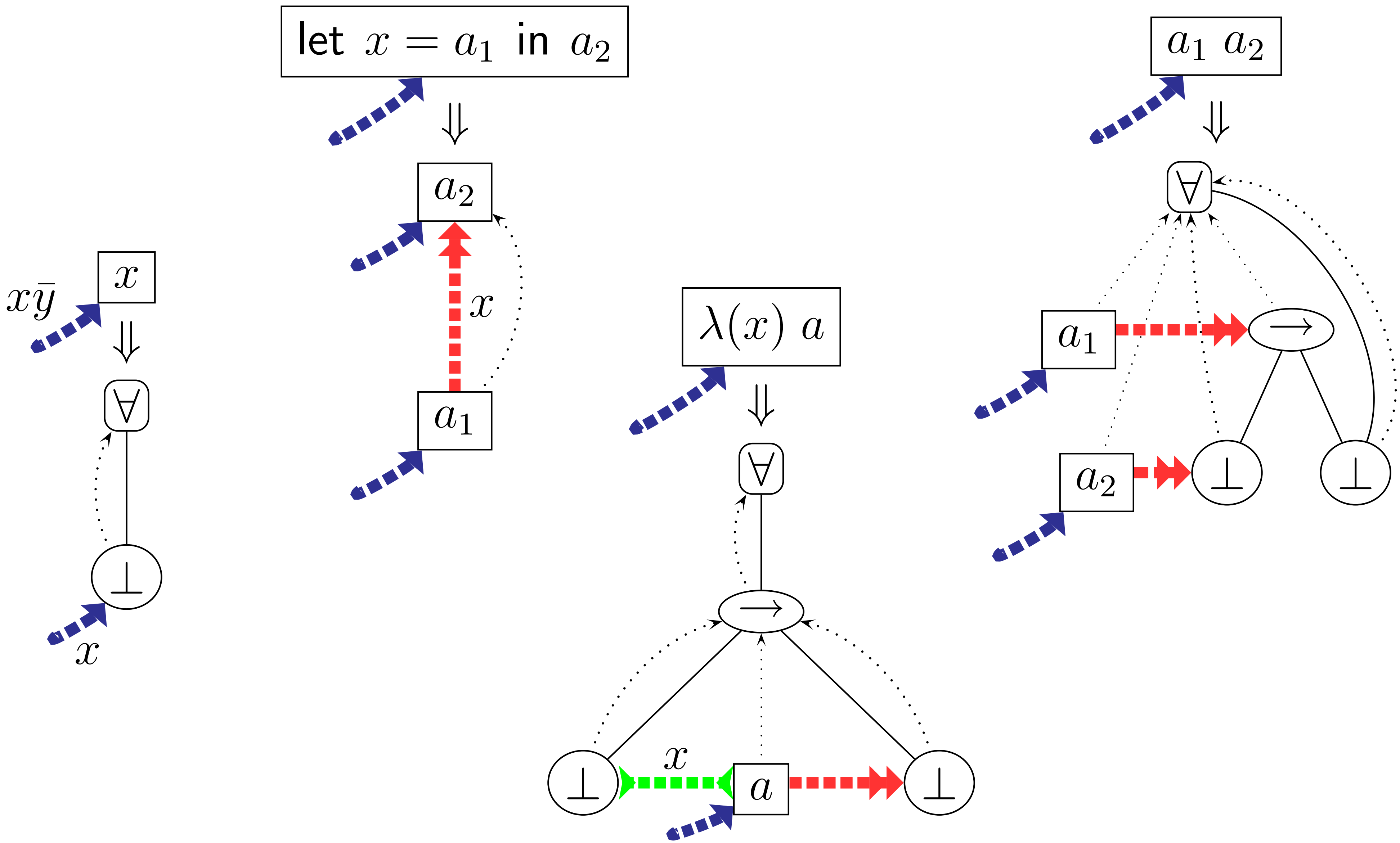
$$s_1 \text{ depends on } s_2 \text{ if } s_2 \dashrightarrow n \text{ and } n \text{ is inner } s_1.$$

A constraint χ is admissible if the dependency relation is acyclic.
(Except for pathological cases, other cases do not have solutions).

Strategy for solving admissible constraints

- Independent schemes may be solved first, by Inst-Elim-Poly
- The unification edge that is introduced may be solved immediately.
- This way, no constaint edge is ever duplicated.

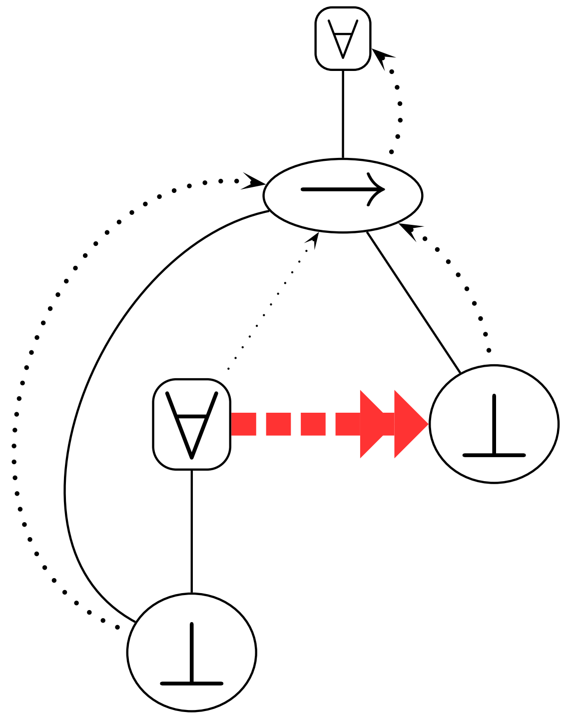
Typing constraints



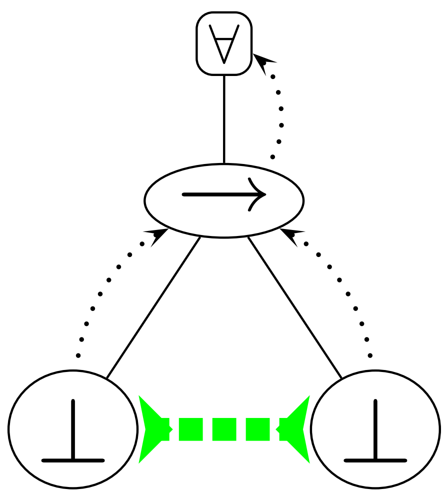
$$\boxed{\lambda(x)\ x}$$

Typing example $\lambda(x)\ x$

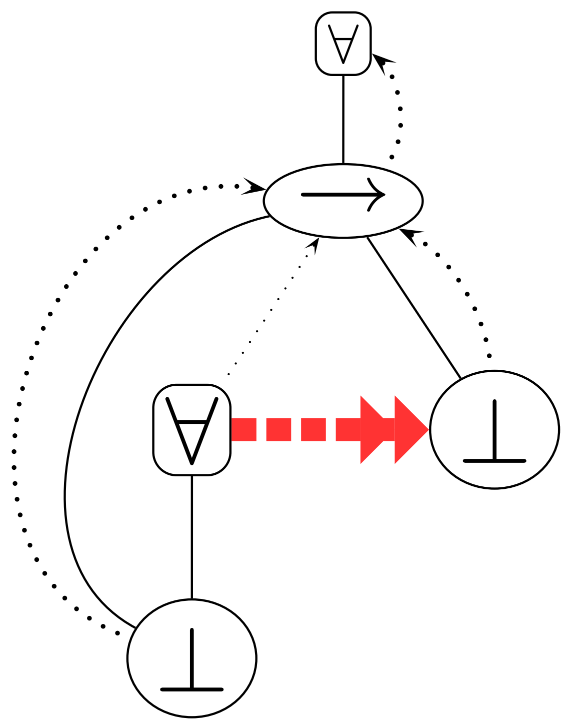
$\boxed{\lambda(x)\ x}$ $\Rightarrow$

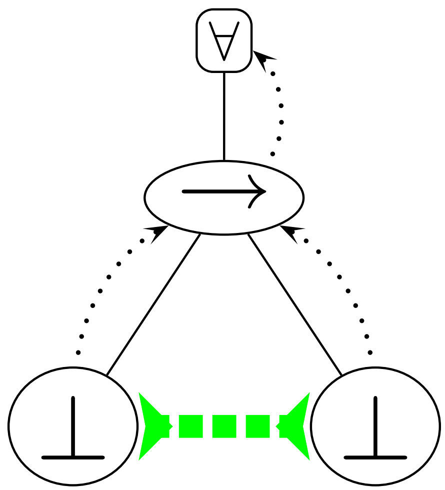


Typing example $\lambda(x)\ x$

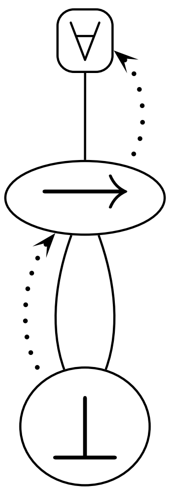


$\Longleftarrow$





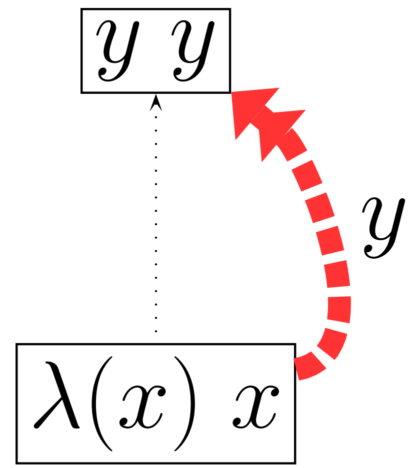
$\Rightarrow$



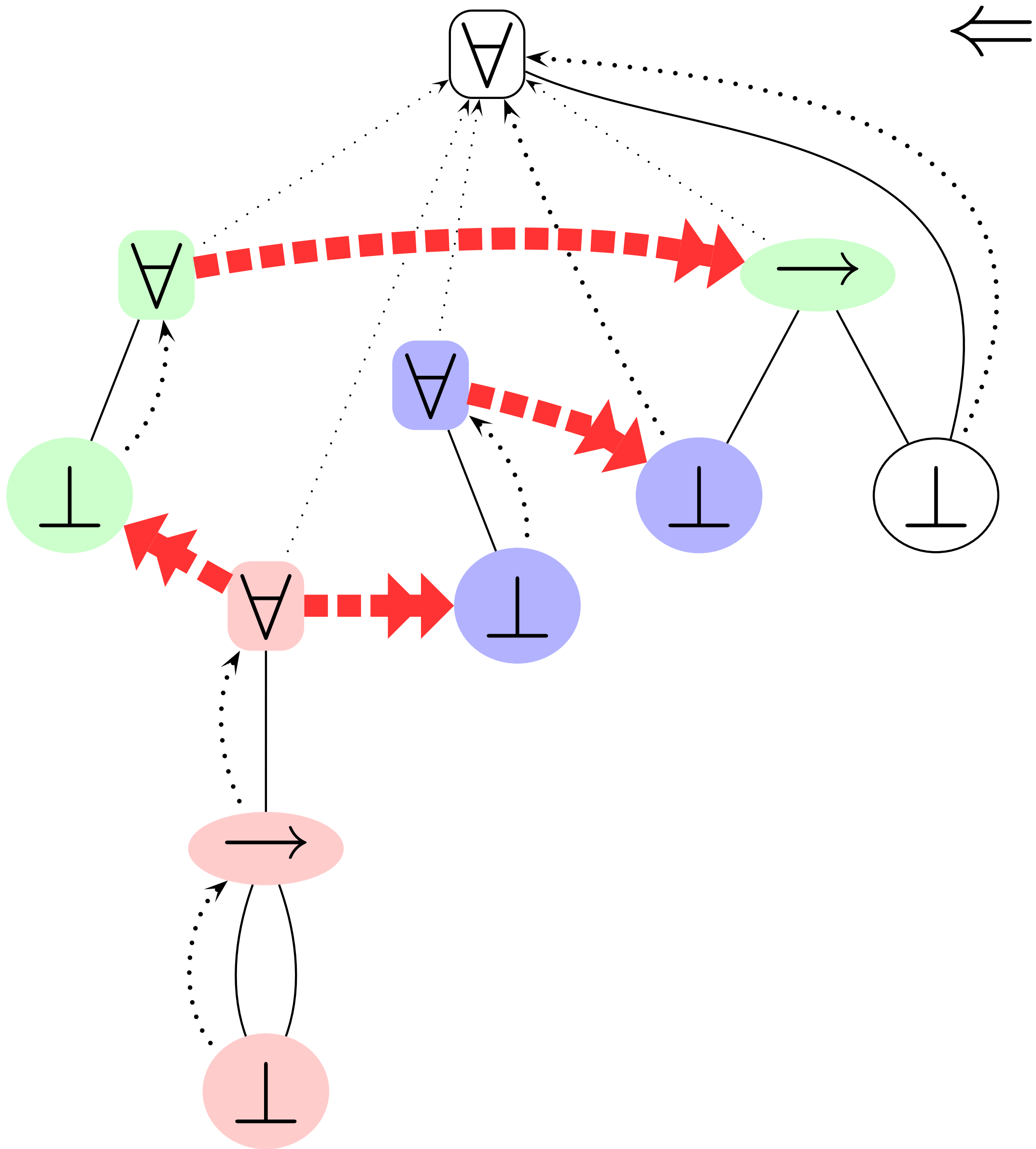
Example $\mathsf{let}\ y = \lambda(x)\ x\ \mathsf{in}\ y\ y$

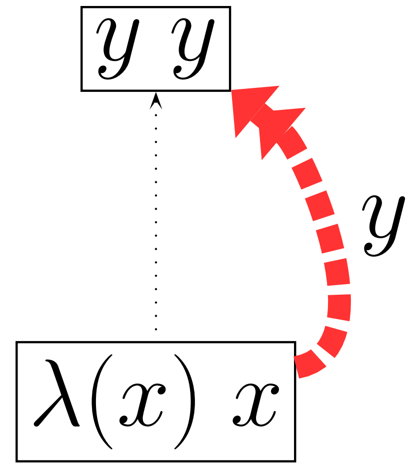
$$\boxed{\begin{array}{c} \mathsf{let}\ y = \lambda(x)\ x \\ \mathsf{in}\ y\ y \end{array}}$$

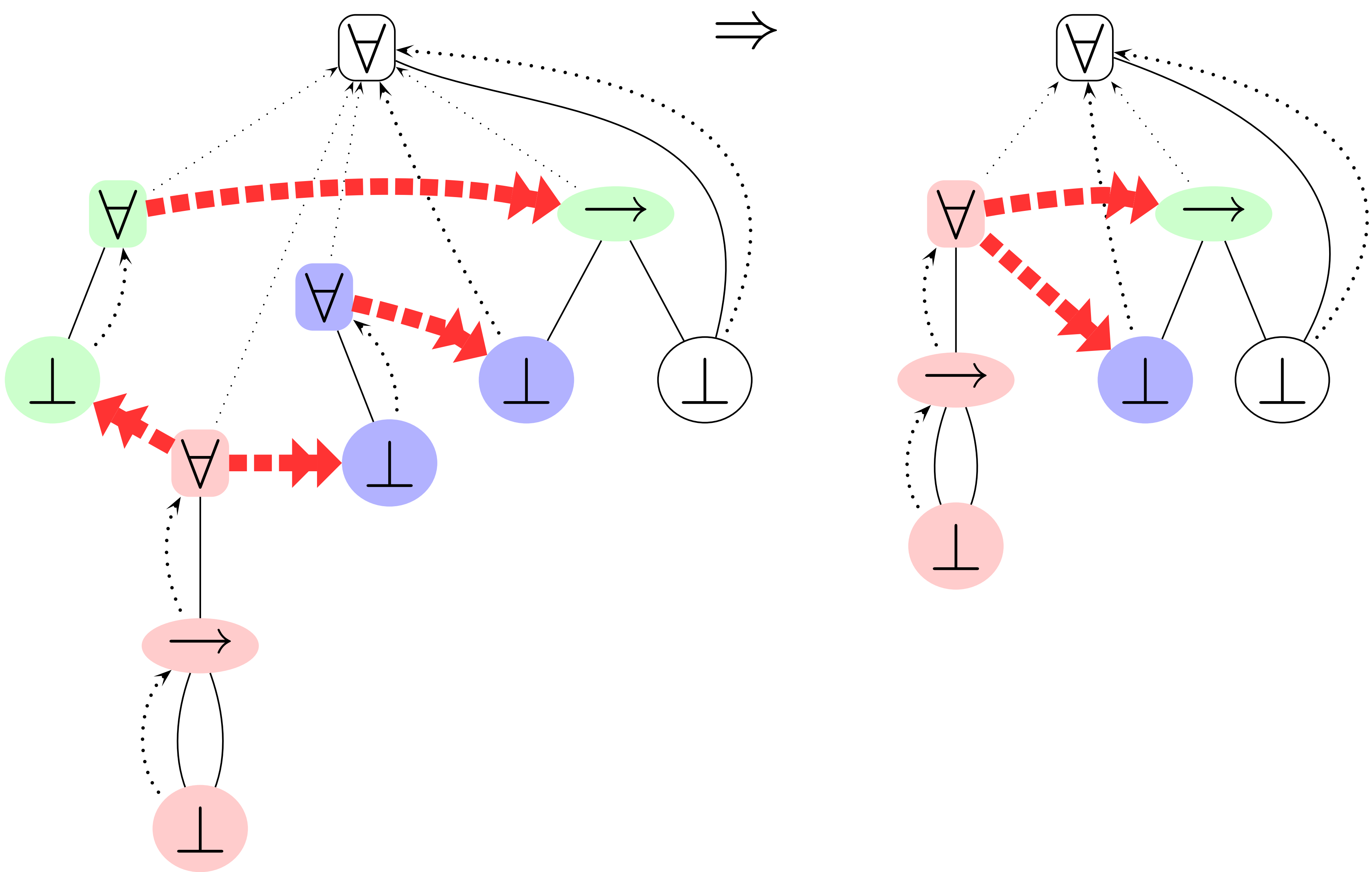
$$\boxed{\begin{array}{c}\textsf{let } y = \lambda(x)\ x \\ \textsf{in } y\ y\end{array}} \Rightarrow$$



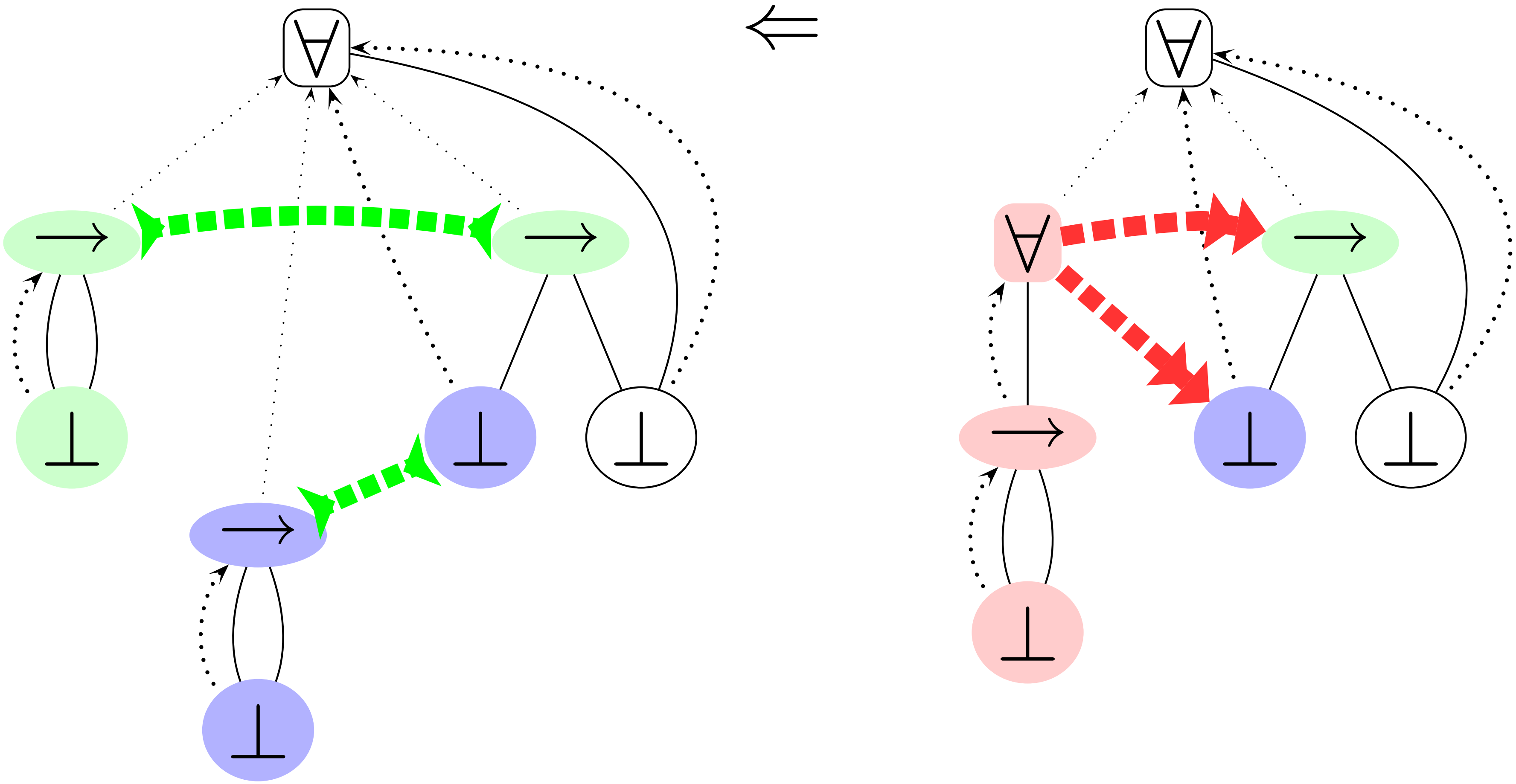
Example let $y = \lambda(x)\ x$ in $y\ y$



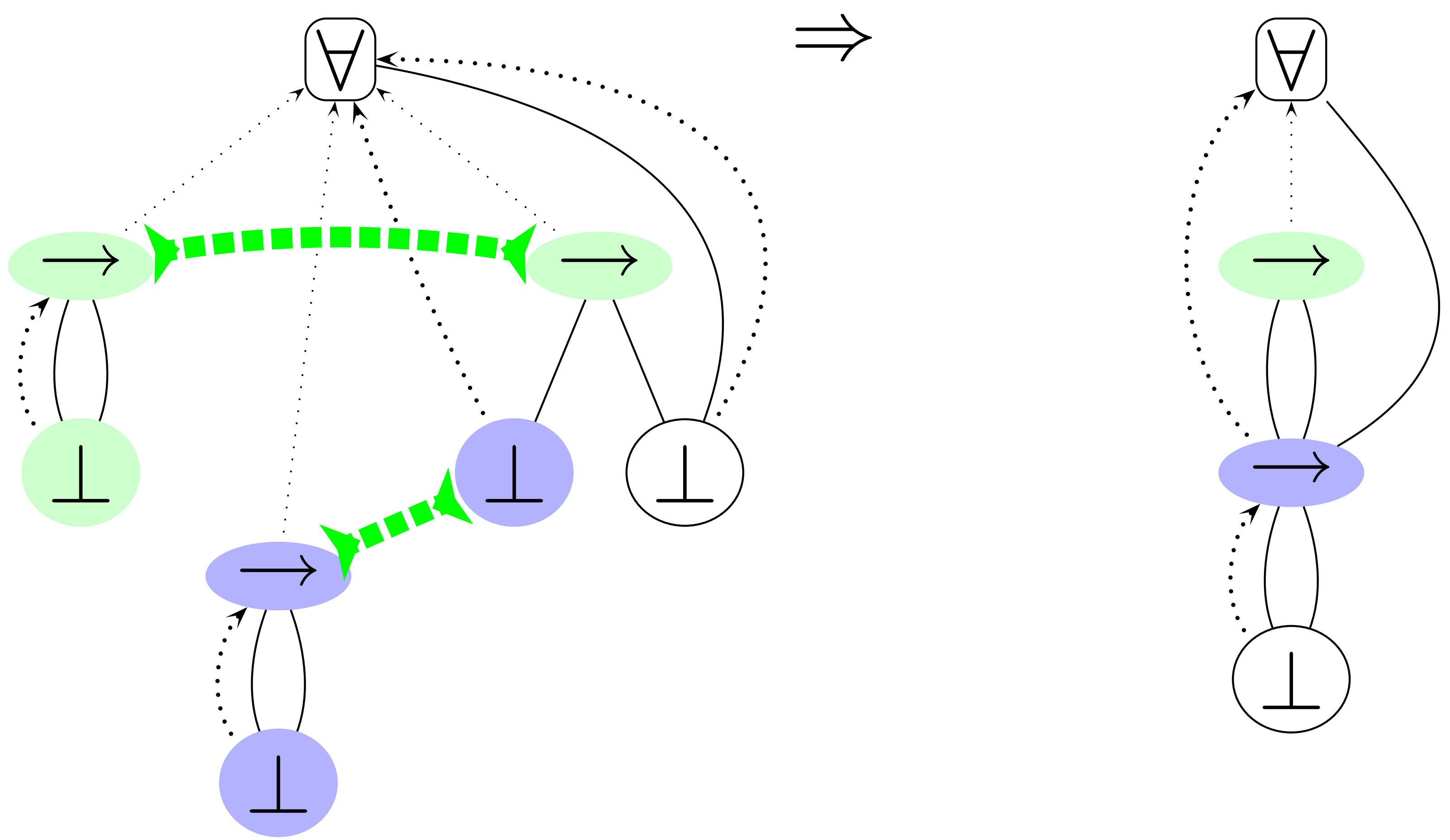




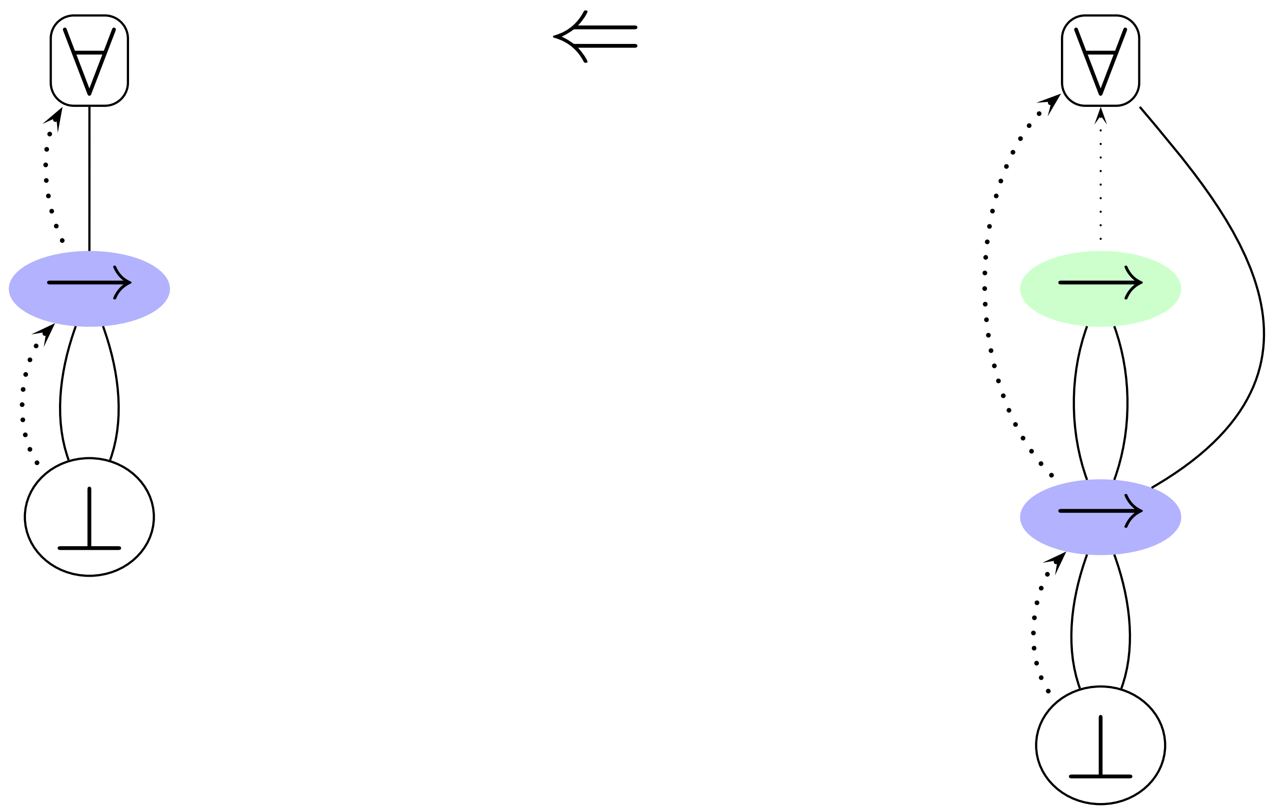
Example **let** $y = \lambda(x)\ x$ **in** $y\ y$



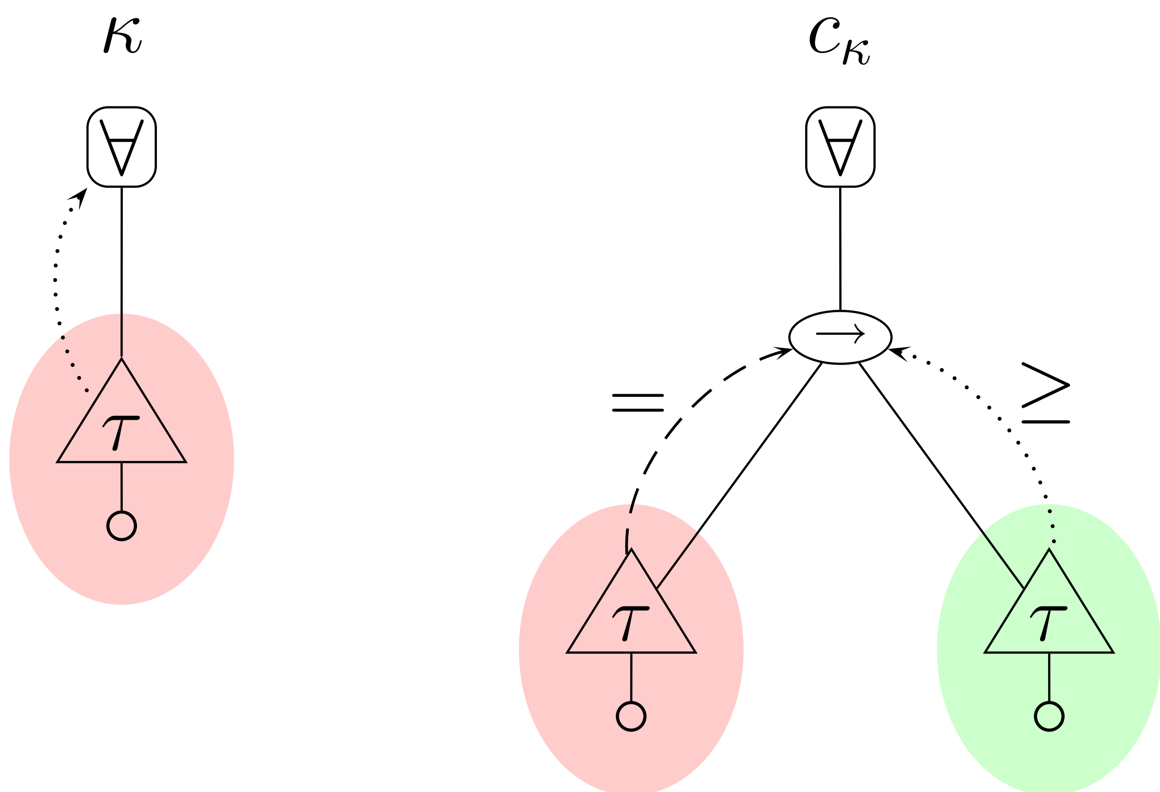
Example **let** $y = \lambda(x)\ x$ **in** $y\ y$



Example **let** $y = \lambda(x)\ x$ **in** $y\ y$



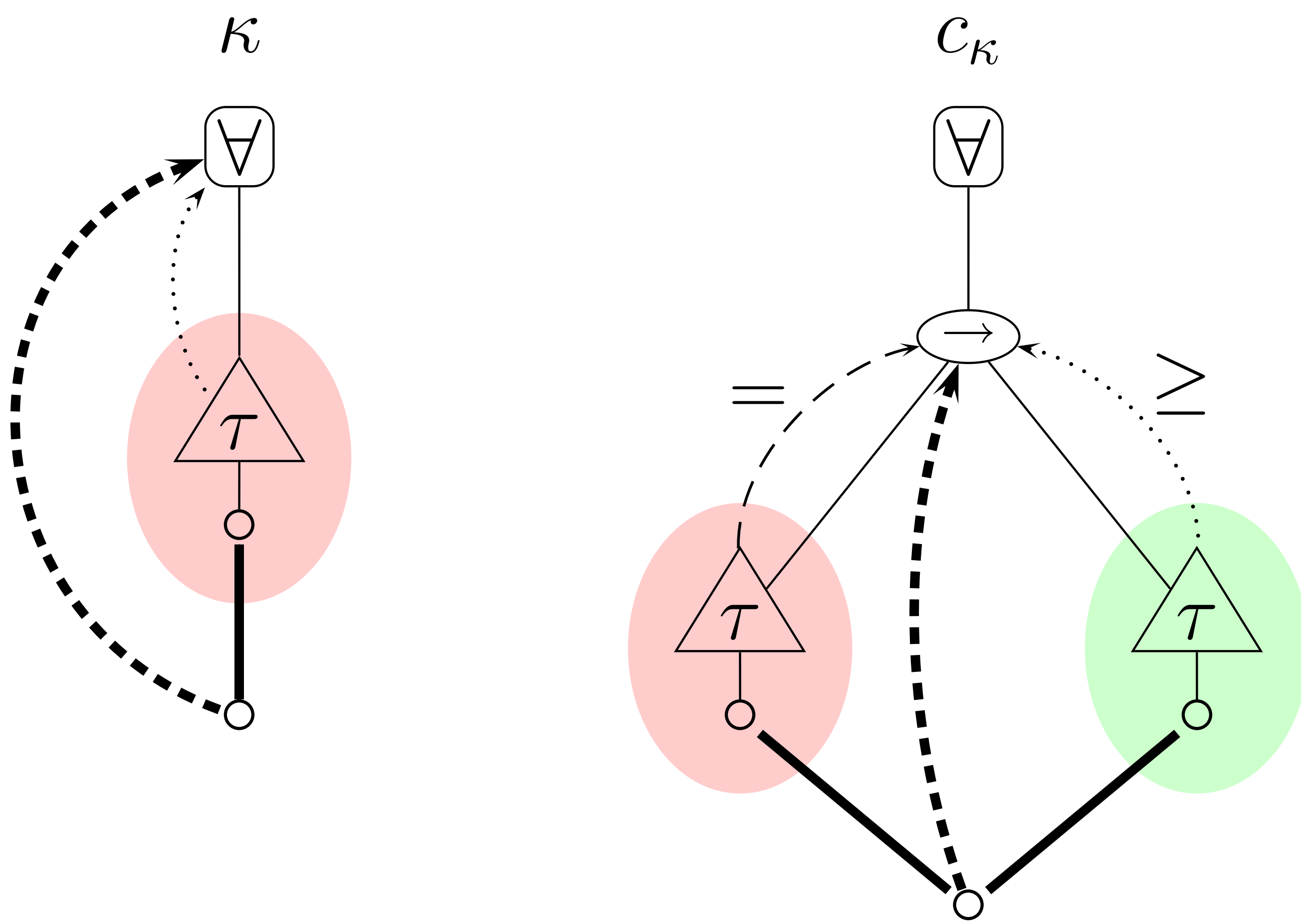
Coercions: $(a : \kappa)$ is typed as $c_\kappa\ a$



$\forall(\alpha)\,\tau$

$\forall(\gamma = \forall(\alpha)\,\tau)\,\forall(\gamma' > \forall(\alpha)\,\tau)\,\gamma \rightarrow \gamma'$

Coercions: $(a : \kappa)$ is typed as $c_\kappa\ a$



$\exists\bar\beta\forall(\alpha)\,\tau$ $\qquad$ $\forall(\bar\beta)\,\forall(\gamma = \forall(\alpha)\,\tau)\,\forall(\gamma' > \forall(\alpha)\,\tau)\,\gamma \rightarrow \gamma'$

Type soundness

We show type soundness in $\mathsf{IML}^{\mathsf{F}}$ a fully implicitly typed version of $\mathsf{XML}^{\mathsf{F}}$.

Type soundness

We show type soundness in $\mathrm{IML}^{\mathsf{F}}$ a fully implicitly typed version of $\mathrm{XML}^{\mathsf{F}}$.

$\mathrm{XML}^{\mathsf{F}}$ is defined as $\mathrm{IML}^{\mathsf{F}}$, replacing $\sqsubseteq$ by $\sqsubseteq^{\boxminus}$ everywhere where $\sqsubseteq^{\boxminus}$ is $(\sqsubseteq \cup \sqsupset)^*$

No type inference and no principal types in $\mathrm{IML}^{\mathsf{F}}$

This changes the semantics of contraints, which have more solutions. Entailment is incomparable.

Transformations rules of $\mathrm{XML}^{\mathsf{F}}$ are not not complete or not sound in $\mathrm{IML}^{\mathsf{F}}$

Type soundness

Subject reduction means that $\longrightarrow$ is a subrelation of $\boxed{\cdot} \Vdash \boxed{\cdot}$.

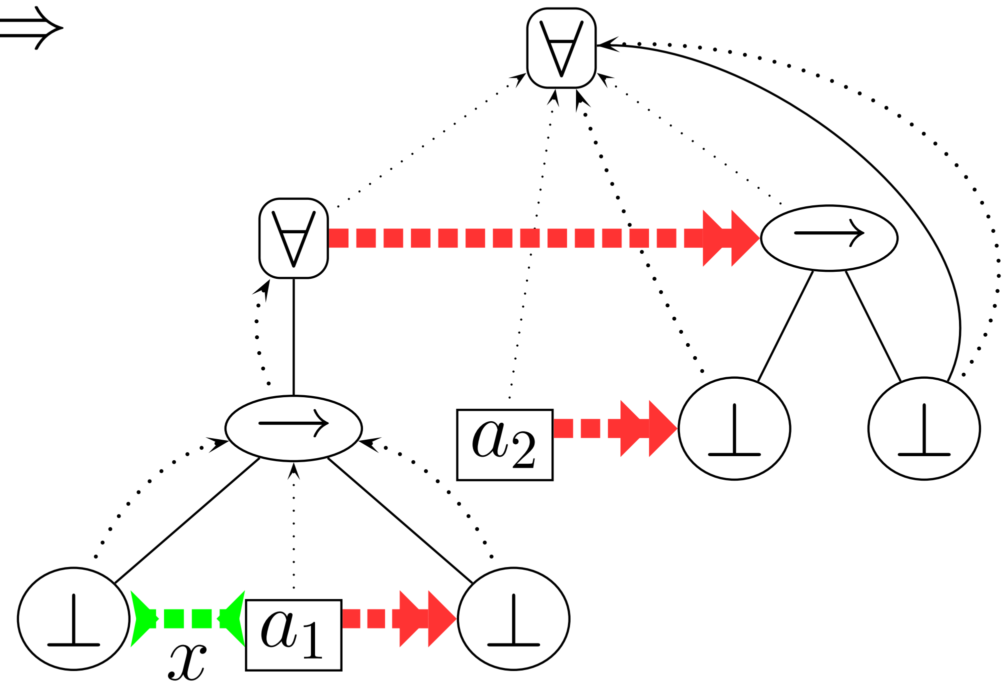
We show that $\boxed{\cdot} \Vdash \boxed{\cdot}$ satisfies the rules defining $\longrightarrow$.

Progress is easy.

β preserves typings

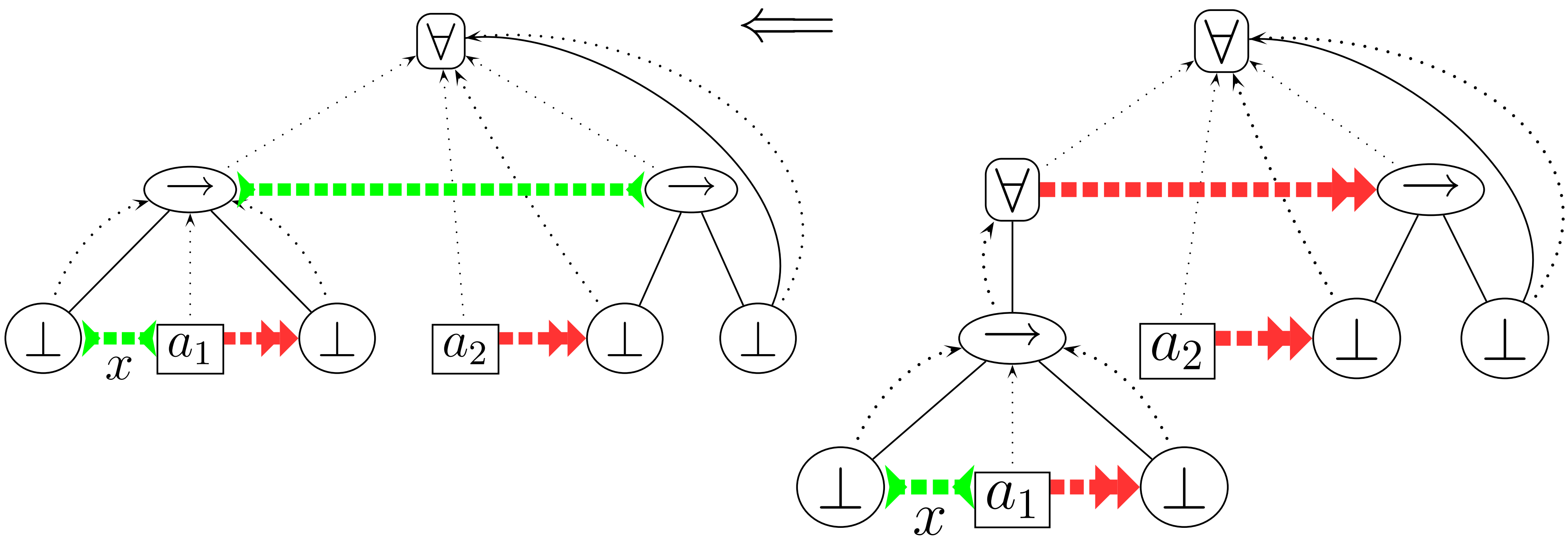
$$\boxed{(\lambda(x)\ a_1)\ a_2}$$

$\boxed{(\lambda(x)\ a_1)\ a_2}$ $\Longrightarrow$



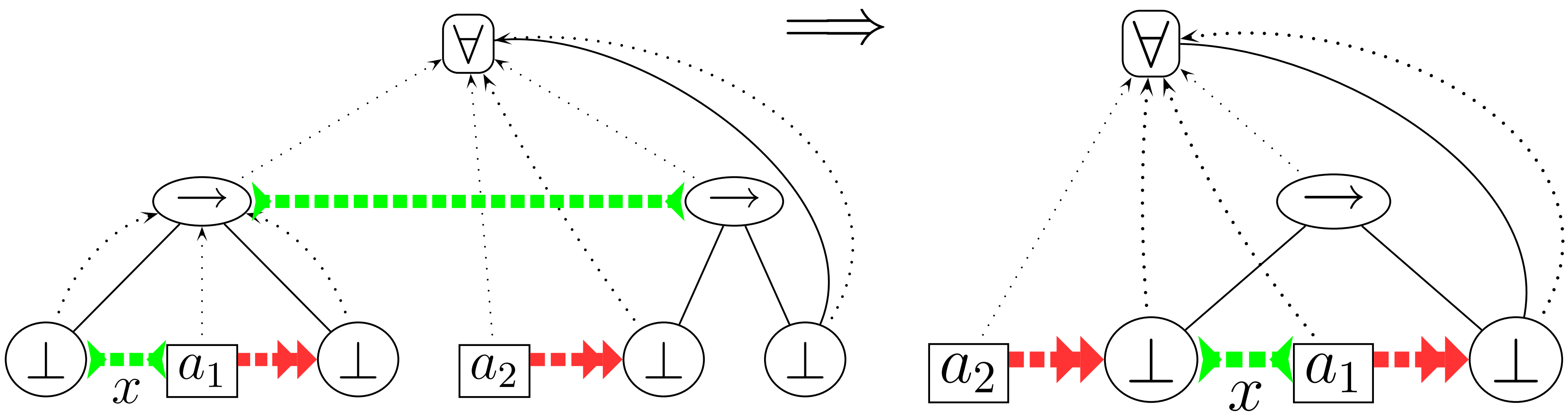
Equivalence, by definition

β preserves typings

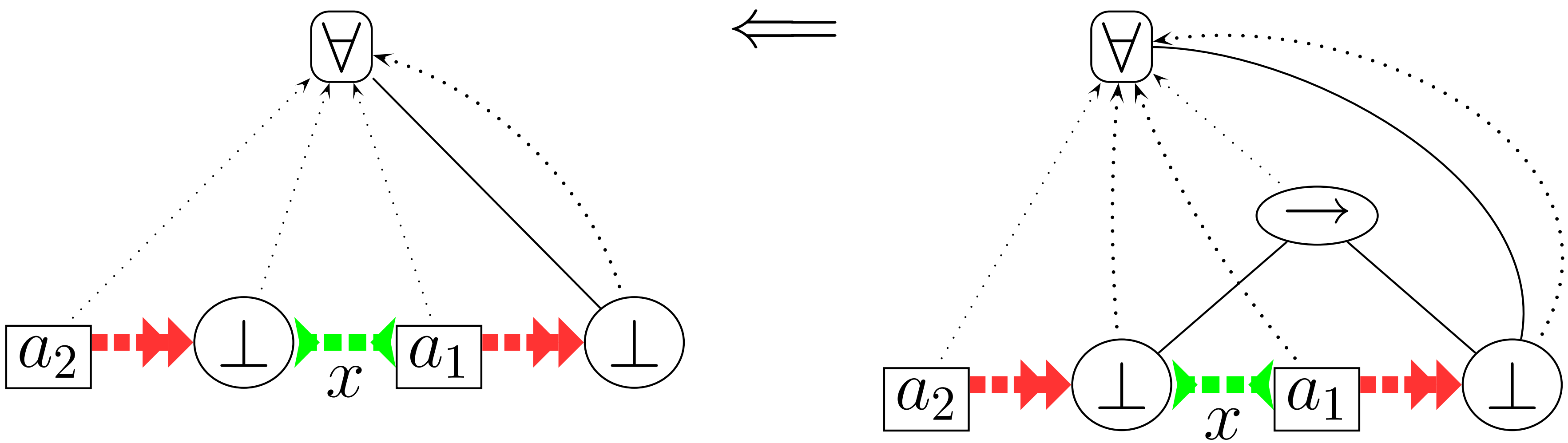


Entailment, †

β preserves typings

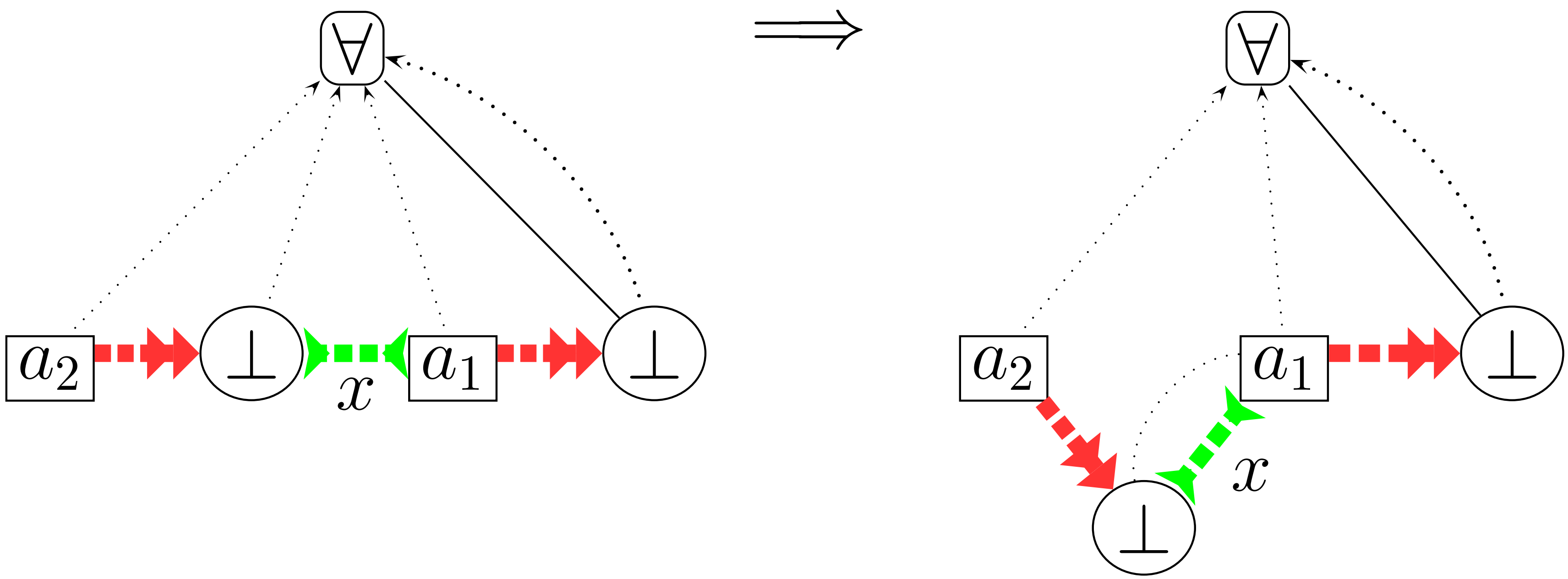


β preserves typings

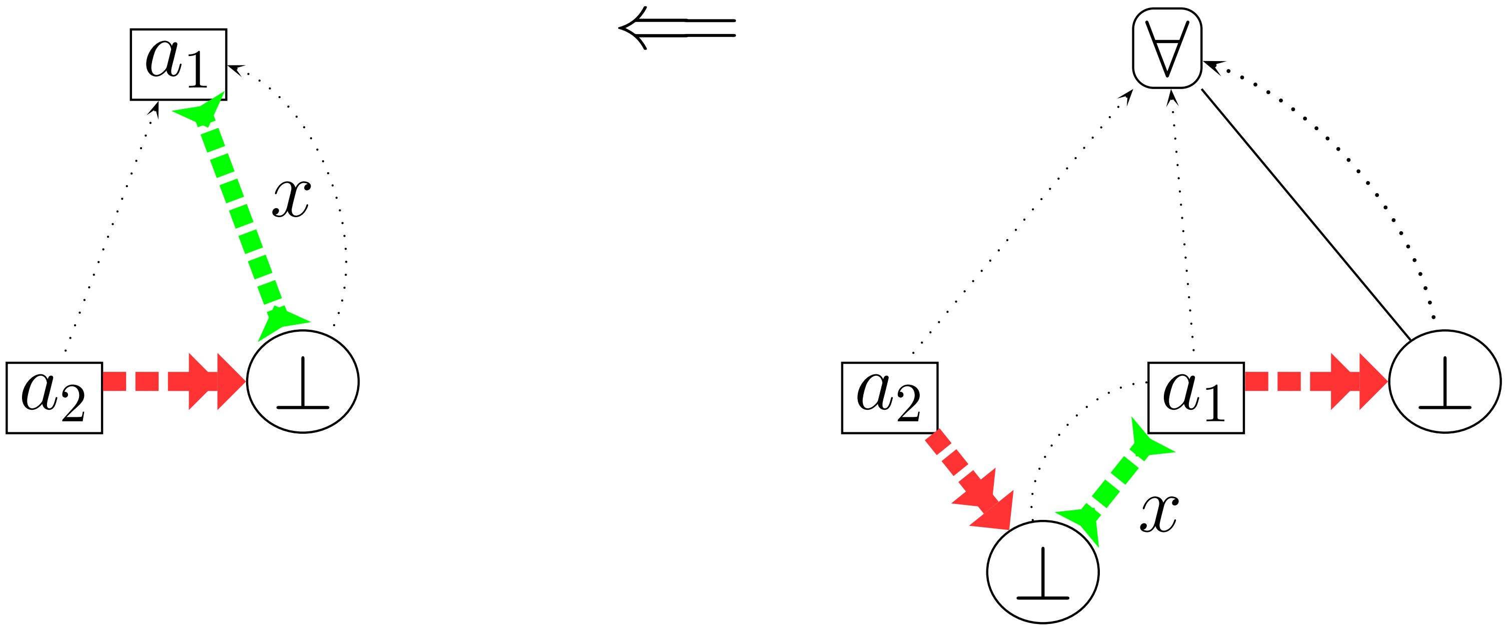


Equivalence, by existential elimination

β preserves typings

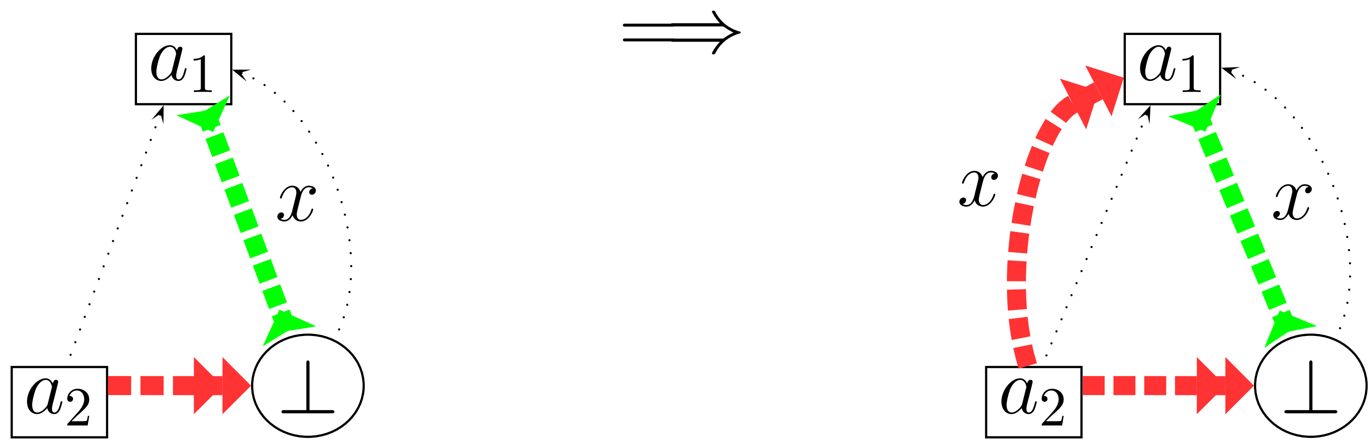


Entailment, by inverse instance



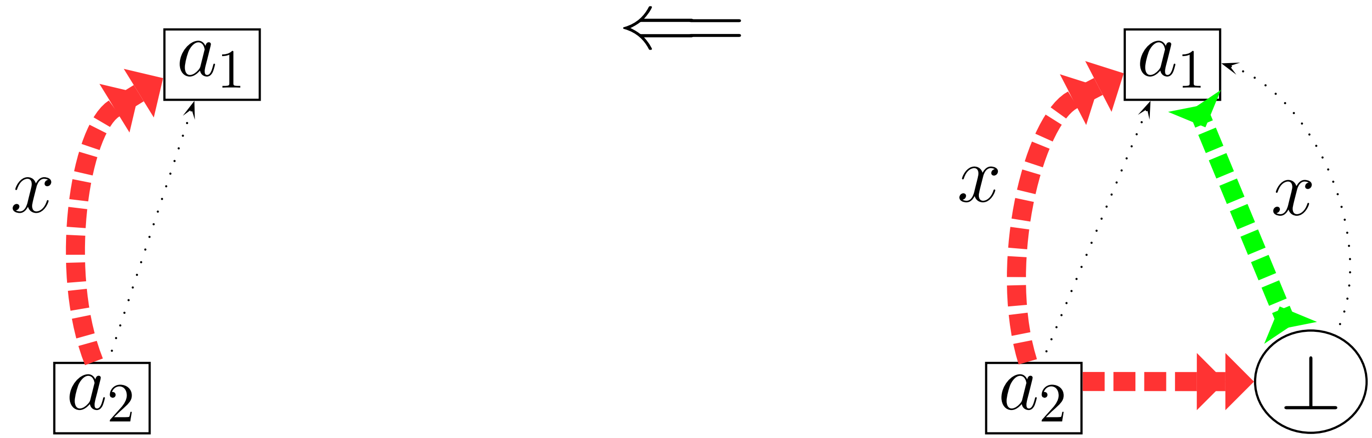
Entailment, by Inst-Bot, †

β preserves typings

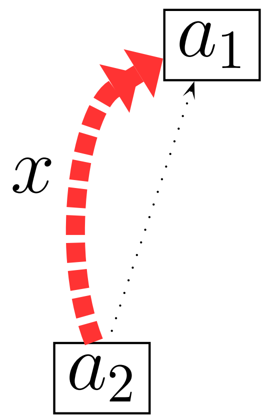


Equivalence, decomposition of $\sqsubseteq^{\boxminus}$

β preserves typings



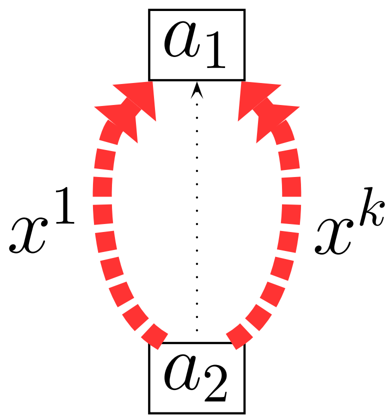
Entailment, just dropping constraints



$\Longrightarrow$ $\boxed{\text{let } x = a_1 \text{ in } a_2}$

Equivalence, by definition

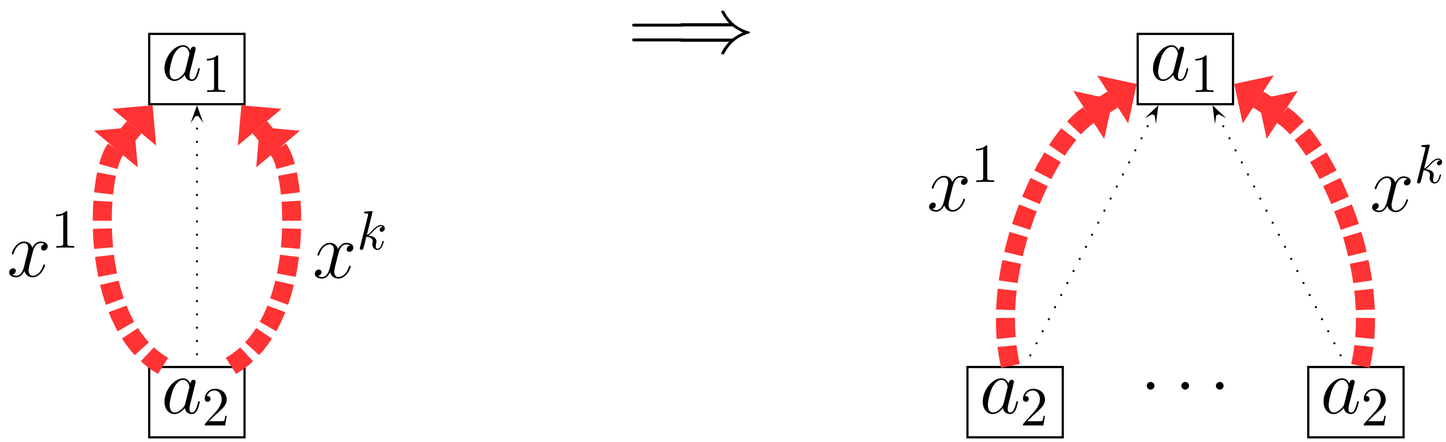
β preserves typings



$\Longleftarrow$ $\boxed{\mathsf{let}\ x = a_1\ \mathsf{in}\ a_2}$

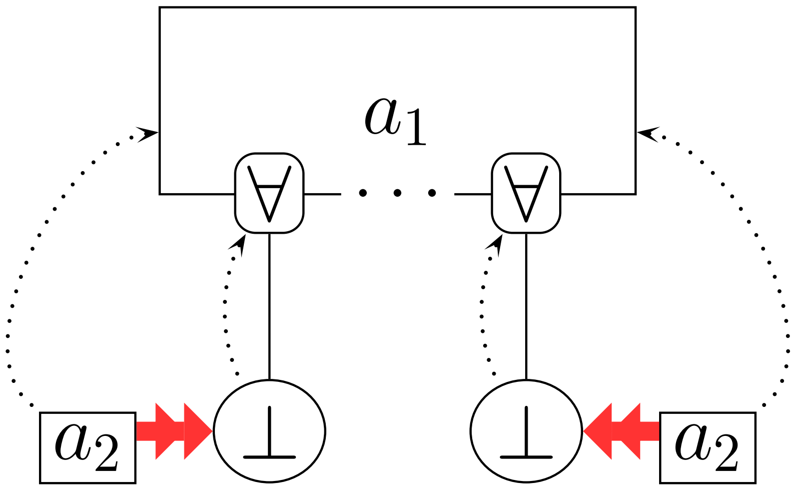
Equivalence, by definition

β preserves typings

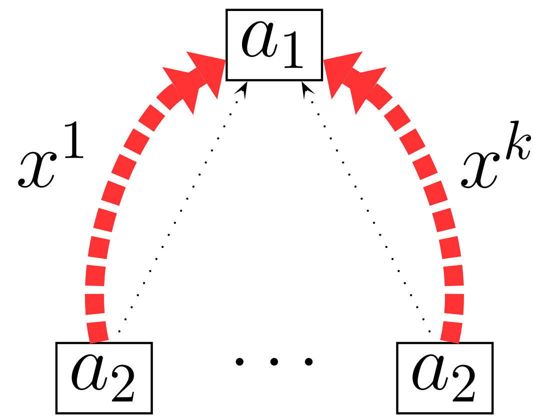


$$\left(\begin{array}{c} \Vdash \\ \not\dashV \end{array}\right)$$

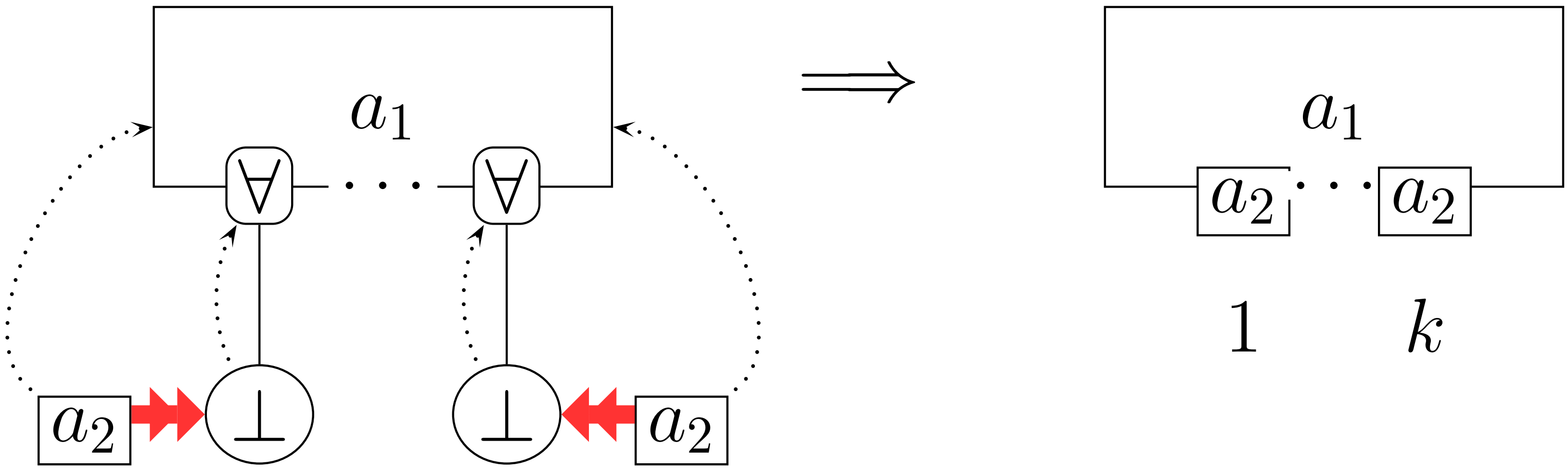
Entailment, by Inst-Copy, †



$\Longleftarrow$

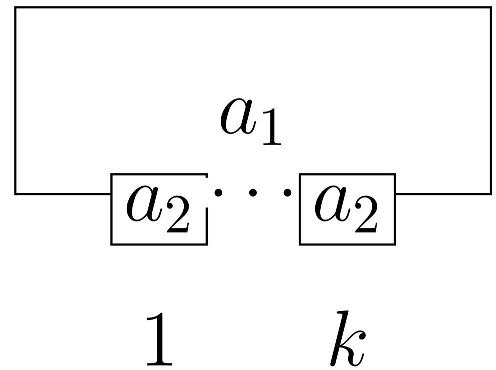


Equivalence, zooing on details.



Entailment, by Inst-Bot

$$\boxed{a_1[a_2/x]} \quad \Longleftarrow$$



Equivalence, by definition,

Conclusions

- Simpler, canonical definition of ML^{F}.
- Efficient, scalable type inference.
- Generalizing type constraint for ML.
  Makes type inference independent of the underlying language.
- Good basis for further extensions:
  higher-order types, recusive types, existential types, ...
- Also to be explored: semi-unification problem for ML^{F} types.

See http://gallium.inria.fr/~remy/mlf/

Appendices

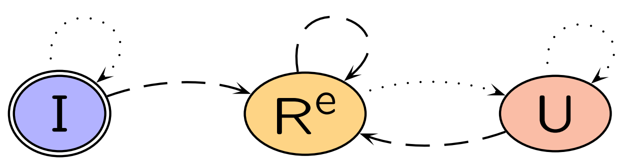
Permissions

Nodes/contexts are *partitioned* into four categories:

I	*irreversible*
R^{e}	*explicitly reversible*
R^{i}	*implicitly reversible*
U	*unsafe*.

They are uniquely determined by the binding tree.

- R^{i}-nodes are non-bottom nodes whose incoming binding edges all originate from other R^{i}-nodes.
- The remaining nodes are further classified by looking at the sequence of labels obtained from following their binding edges in the inverse direction (starting from the root) in the automaton drawn here.



Permissions

Relations

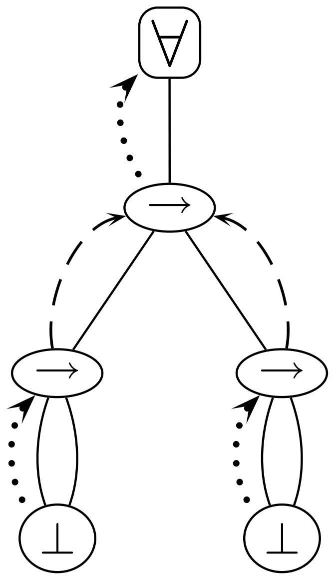
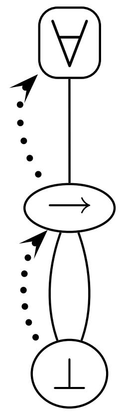
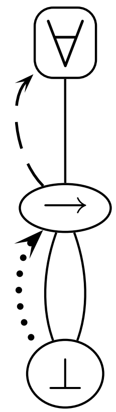
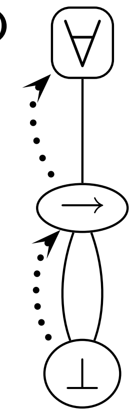
		Grafting	Merging	Raising	Weakening
Instance	$\sqsubseteq$	I	I, R	I, R	I
Abstraction	$\sqsubseteq$	–	R	R	–
Similarity	$\preccurlyeq$	–	R^i	R^i	–

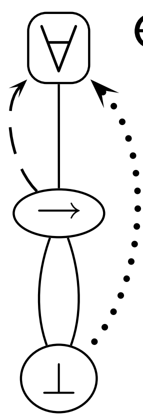
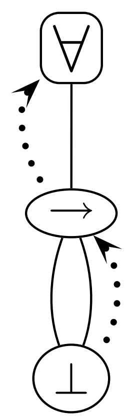
Decompositions

We may always treat types up to $\preccurlyeq$, since $(\sqsubseteq \cup \succcurlyeq)^* = (\sqsubseteq; \succcurlyeq)$

We may also (sometimes) treat types up to $\sqsubseteq$, since $(\sqsubseteq \cup \sqsupseteq)^* = (\sqsubseteq; \sqsupseteq)$

Reset:

Applying  to  returns  equivalent to 

Similarly,  equivalent to 

See Gen typing rule.

Recasting ML

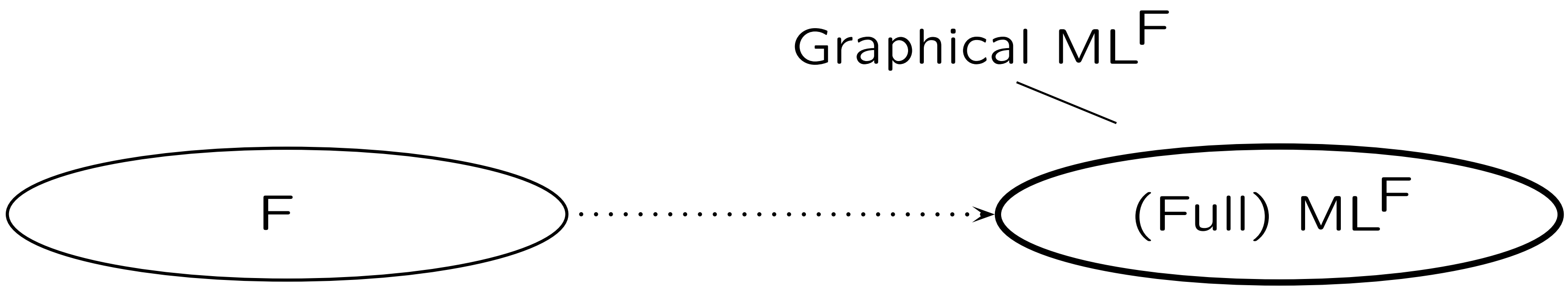
A revisited syntactic presentation of ML$^{\mathsf{F}}$, with an interpretation of types as sets of System-F types.

- It justifies the choice of types and type instance.
- We exhibit a correspondance with implicitly typed and explicitly typed versions of ML$^{\mathsf{F}}$
- We encodes ML$^{\mathsf{F}}$ into F$^{\mathsf{let}}$ (an extension of F with intersection types)

The instance relation is (slightly) enhanced by correcting artifacts of the syntactic definition in the original relation.

However, this presentation is restricted to Plain ML$^{\mathsf{F}}$ (types are stratified).

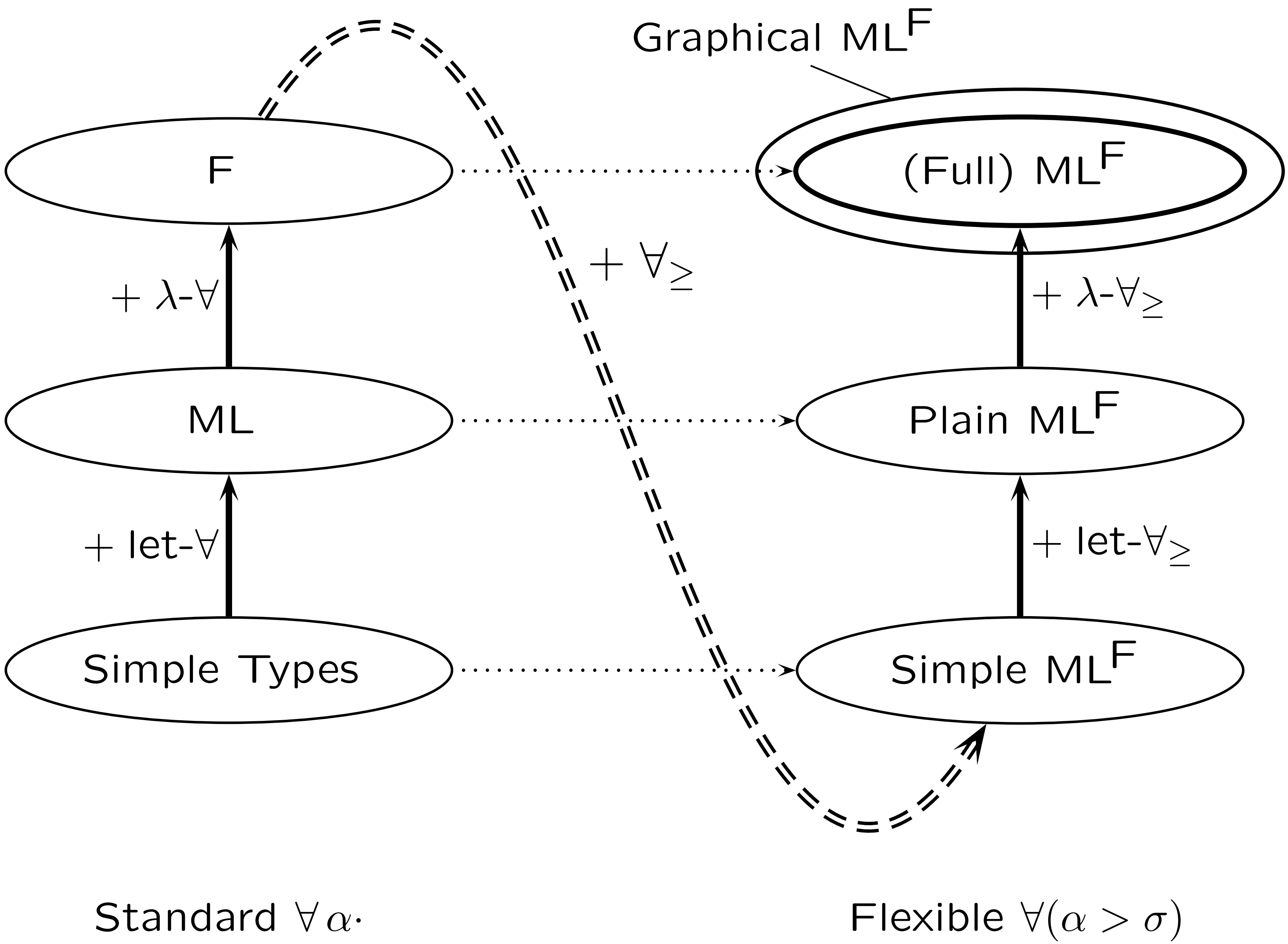
A family of languages



Standard $\forall\,\alpha\cdot$

Flexible $\forall(\alpha > \sigma)$

A family of languages



Syntactic instance

Type Equivalence

Eq-Refl
$$(Q)\ \sigma \equiv \sigma$$

Eq-Trans
$$\frac{(Q)\ \sigma_1 \equiv \sigma_2 \qquad (Q)\ \sigma_2 \equiv \sigma_3}{(Q)\ \sigma_1 \equiv \sigma_3}$$

Eq-Context-R
$$\frac{(Q, \alpha \diamond \sigma)\ \sigma_1 \equiv \sigma_2}{(Q)\ \forall (\alpha \diamond \sigma)\ \sigma_1 \equiv \forall (\alpha \diamond \sigma)\ \sigma_2}$$

Eq-Context-L
$$\frac{(Q)\ \sigma_1 \equiv \sigma_2}{(Q)\ \forall (\alpha \diamond \sigma_1)\ \sigma \equiv \forall (\alpha \diamond \sigma_2)\ \sigma}$$

Eq-Free
$$\frac{\alpha \notin \mathsf{ftv}(\sigma_1)}{(Q)\ \forall (\alpha \diamond \sigma)\ \sigma_1 \equiv \sigma_1}$$

Eq-Comm
$$\frac{\alpha_1 \notin \mathsf{ftv}(\sigma_2) \qquad \alpha_2 \notin \mathsf{ftv}(\sigma_1)}{(Q)\ \forall (\alpha_1 \diamond_1 \sigma_1)\ \forall (\alpha_2 \diamond_2 \sigma_2)\ \sigma \equiv \forall (\alpha_2 \diamond_2 \sigma_2)\ \forall (\alpha_1 \diamond_1 \sigma_1)\ \sigma}$$

Eq-Var
$$(Q)\ \forall (\alpha \diamond \sigma)\ \alpha \equiv \sigma$$

Eq-Mono
$$\frac{(\alpha \diamond \sigma_0) \in Q \qquad (Q)\ \sigma_0 \equiv \tau_0}{(Q)\ \tau \equiv \tau[\tau_0/\alpha]}$$

Type Abstraction

A-Equiv
$$\frac{(Q)\ \sigma_1 \equiv \sigma_2}{(Q)\ \sigma_1 \sqsubseteq \sigma_2}$$

A-Trans
$$\frac{(Q)\ \sigma_1 \sqsubseteq \sigma_2 \qquad (Q)\ \sigma_2 \sqsubseteq \sigma_3}{(Q)\ \sigma_1 \sqsubseteq \sigma_3}$$

A-Context-R
$$\frac{(Q, \alpha \diamond \sigma)\ \sigma_1 \sqsubseteq \sigma_2}{(Q)\ \forall (\alpha \diamond \sigma)\ \sigma_1 \sqsubseteq \forall (\alpha \diamond \sigma)\ \sigma_2}$$

A-Hyp
$$\frac{(\alpha_1 = \sigma_1) \in Q}{(Q)\ \sigma_1 \sqsubseteq \alpha_1}$$

A-Context-L
$$\frac{(Q)\ \sigma_1 \sqsubseteq \sigma_2}{(Q)\ \forall (\alpha = \sigma_1)\ \sigma \sqsubseteq \forall (\alpha = \sigma_2)\ \sigma}$$

Type Instance

I-Abstract
$$\frac{(Q)\ \sigma_1 \sqsubseteq \sigma_2}{(Q)\ \sigma_1 \leqslant \sigma_2}$$

I-Trans
$$\frac{(Q)\ \sigma_1 \leqslant \sigma_2 \qquad (Q)\ \sigma_2 \leqslant \sigma_3}{(Q)\ \sigma_1 \leqslant \sigma_3}$$

I-Context-R
$$\frac{(Q, \alpha \diamond \sigma)\ \sigma_1 \leqslant \sigma_2}{(Q)\ \forall (\alpha \diamond \sigma)\ \sigma_1 \leqslant \forall (\alpha \diamond \sigma)\ \sigma_2}$$

I-Hyp
$$\frac{(\alpha_1 \geq \sigma_1) \in Q}{(Q)\ \sigma_1 \leqslant \alpha_1}$$

I-Context-L
$$\frac{(Q)\ \sigma_1 \leqslant \sigma_2}{(Q)\ \forall (\alpha > \sigma_1)\ \sigma \leqslant \forall (\alpha > \sigma_2)\ \sigma}$$

I-Bot
$$(Q)\ \bot \leqslant \sigma$$

I-Rigid
$$\frac{}{(Q)\ \forall (\alpha > \sigma_1)\ \sigma \leqslant \forall (\alpha = \sigma_1)\ \sigma}$$

Type Equivalence

Eq-Trans

$$\frac{(Q)\ \sigma_1 \equiv \sigma_2 \qquad (Q)\ \sigma_2 \equiv \sigma_3}{(Q)\ \sigma_1 \equiv \sigma_3}$$

Eq-Context-R

$$\frac{(Q, \alpha \diamond \sigma)\ \sigma_1 \equiv \sigma_2}{(Q)\ \forall(\alpha \diamond \sigma)\, \sigma_1 \equiv \forall(\alpha \diamond \sigma)\, \sigma_2}$$

Eq-Refl

$$(Q)\ \sigma \equiv \sigma$$

Eq-Context-L

$$\frac{(Q)\ \sigma_1 \equiv \sigma_2}{(Q)\ \forall(\alpha \diamond \sigma_1)\, \sigma \equiv \forall(\alpha \diamond \sigma_2)\, \sigma}$$

Eq-Free

$$\frac{\alpha \notin \mathsf{ftv}(\sigma_1)}{(Q)\ \forall(\alpha \diamond \sigma)\, \sigma_1 \equiv \sigma_1}$$

Eq-Comm

$$\frac{\alpha_1 \notin \mathsf{ftv}(\sigma_2) \qquad \alpha_2 \notin \mathsf{ftv}(\sigma_1)}{(Q)\ \forall(\alpha_1 \diamond_1 \sigma_1)\, \forall(\alpha_2 \diamond_2 \sigma_2)\, \sigma \equiv \forall(\alpha_2 \diamond_2 \sigma_2)\, \forall(\alpha_1 \diamond_1 \sigma_1)\, \sigma}$$

Eq-Var

$$(Q)\ \forall(\alpha \diamond \sigma)\, \alpha \equiv \sigma$$

Eq-Mono

$$\frac{(\alpha \diamond \sigma_0) \in Q \qquad (Q)\ \sigma_0 \equiv \tau_0}{}$$

Type Abstraction

A-Equiv
$$\frac{(Q)\ \sigma_1 \equiv \sigma_2}{(Q)\ \sigma_1 \sqsubseteq \sigma_2}$$

A-Trans
$$\frac{(Q)\ \sigma_1 \sqsubseteq \sigma_2 \qquad (Q)\ \sigma_2 \sqsubseteq \sigma_3}{(Q)\ \sigma_1 \sqsubseteq \sigma_3}$$

A-Context-R
$$\frac{(Q, \alpha \diamond \sigma)\ \sigma_1 \sqsubseteq \sigma_2}{(Q)\ \forall(\alpha \diamond \sigma)\,\sigma_1 \sqsubseteq \forall(\alpha \diamond \sigma)\,\sigma_2}$$

A-Hyp
$$\frac{(\alpha_1 = \sigma_1) \in Q}{(Q)\ \sigma_1 \sqsubseteq \alpha_1}$$

A-Context-L
$$\frac{(Q)\ \sigma_1 \sqsubseteq \sigma_2}{(Q)\ \forall(\alpha = \sigma_1)\,\sigma \sqsubseteq \forall(\alpha = \sigma_2)\,\sigma}$$

Type Instance

I-Abstract

$$\frac{(Q)\ \sigma_1 \sqsubseteq \sigma_2}{(Q)\ \sigma_1 \leqslant \sigma_2}$$

I-Trans

$$\frac{(Q)\ \sigma_1 \leqslant \sigma_2 \qquad (Q)\ \sigma_2 \leqslant \sigma_3}{(Q)\ \sigma_1 \leqslant \sigma_3}$$

I-Context-R

$$\frac{(Q, \alpha \diamond \sigma)\ \sigma_1 \leqslant \sigma_2}{(Q)\ \forall(\alpha \diamond \sigma)\,\sigma_1 \leqslant \forall(\alpha \diamond \sigma)\,\sigma_2}$$

I-Hyp

$$\frac{(\alpha_1 \geq \sigma_1) \in Q}{(Q)\ \sigma_1 \leqslant \alpha_1}$$

I-Context-L

$$\frac{(Q)\ \sigma_1 \leqslant \sigma_2}{(Q)\ \forall(\alpha > \sigma_1)\,\sigma \leqslant \forall(\alpha > \sigma_2)\,\sigma}$$

I-Bot

$$(Q)\ \bot \leqslant \sigma$$

I-Rigid

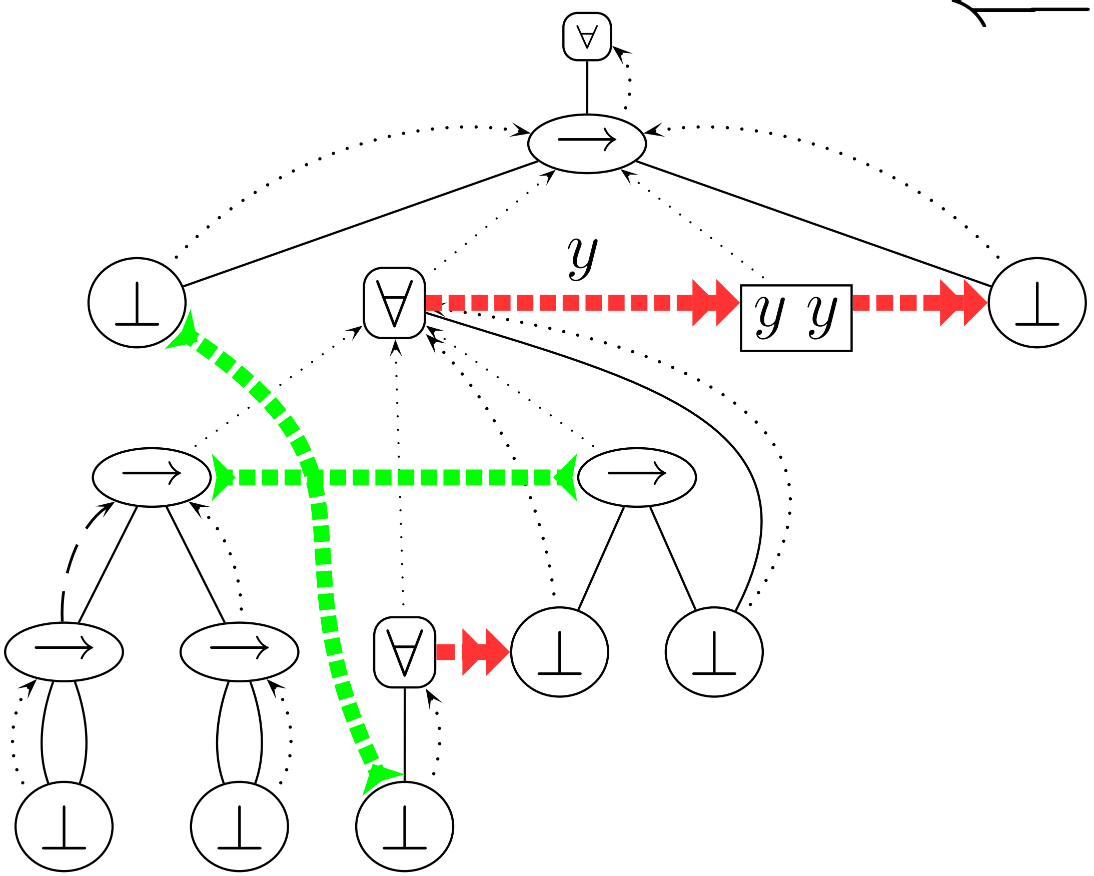
$$\frac{}{(Q)\ \forall(\alpha > \sigma_1)\,\sigma \leqslant \forall(\alpha = \sigma_1)\,\sigma}$$

$$\boxed{\lambda(y : \forall(\alpha)\,\alpha \to \alpha)\; y\; y}$$

(0)

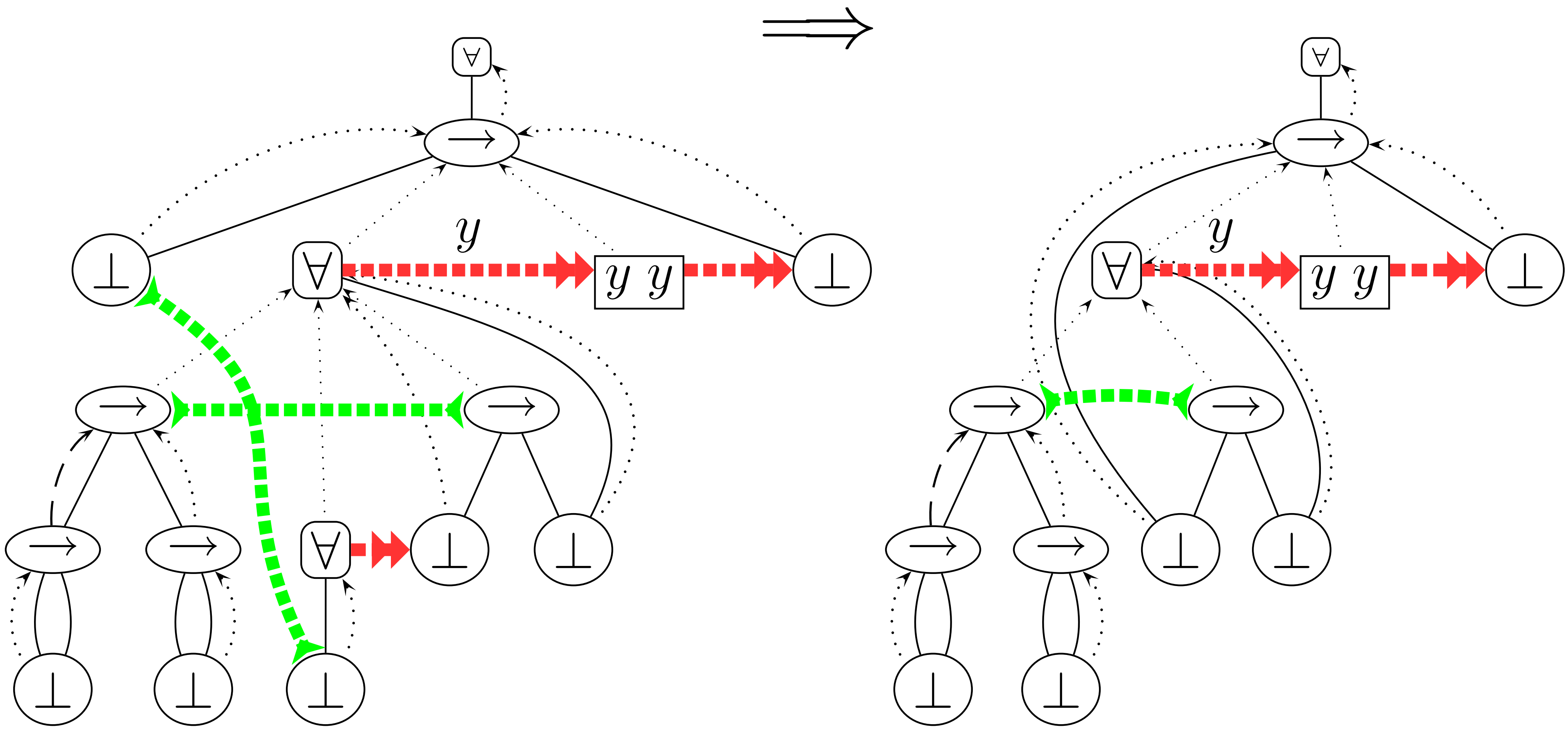
$$\boxed{\lambda(y : \forall(\alpha)\, \alpha \to \alpha)\ y\ y} \quad \Longrightarrow \quad \boxed{\lambda(y)\ \mathsf{let}\ y = c_{\kappa_{\mathsf{id}}}\ y\ \mathsf{in}\ y\ y}$$

(1) by definition



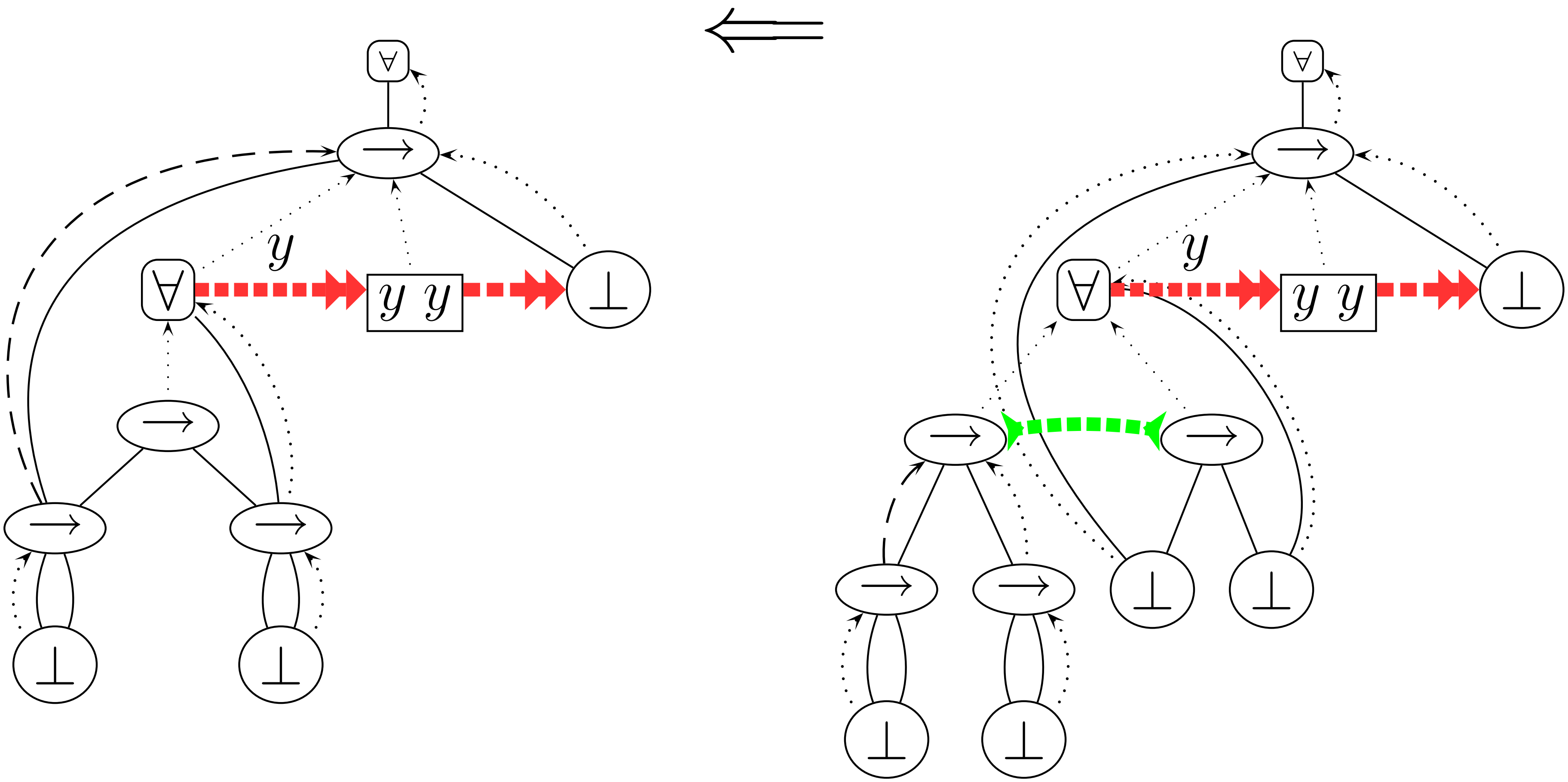
$\Longleftarrow$ $\boxed{\lambda(y)\ \text{let}\ y = c_{\kappa_{\mathsf{id}}}\ y\ \text{in}\ y\ y}$

(2) by definition



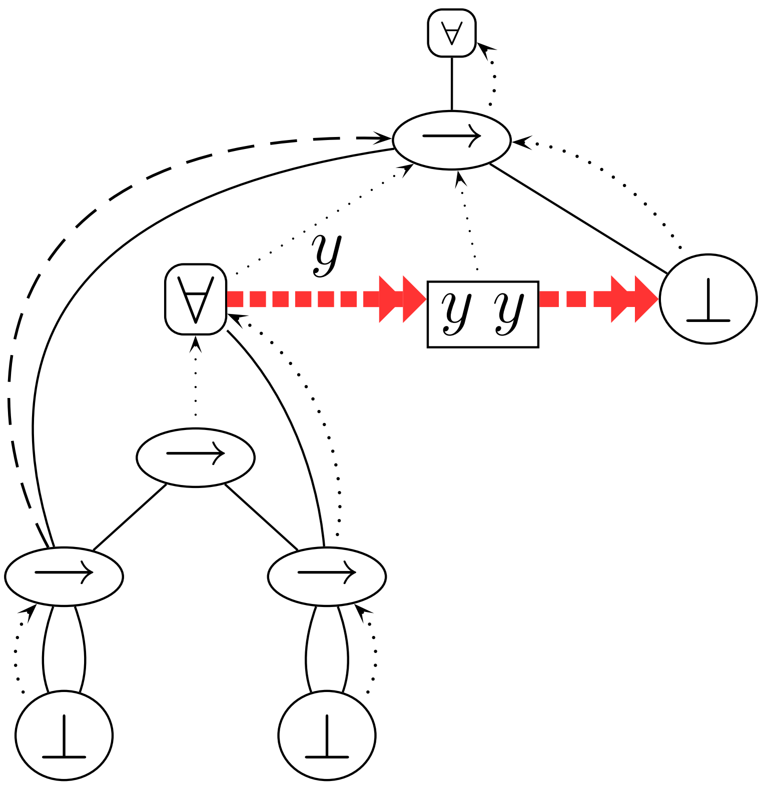
(3) unification

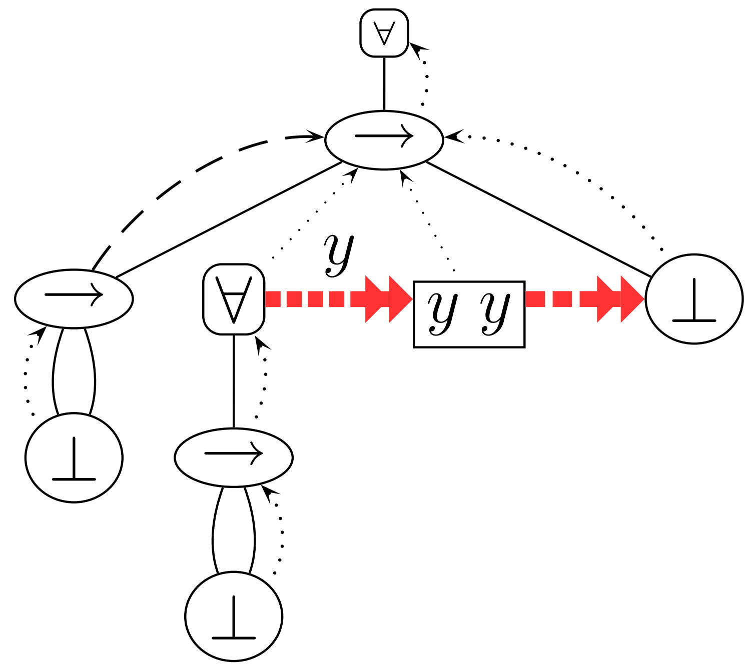
Typing $\lambda(y : \forall(\alpha)\, \alpha \to \alpha)\ y\ y$



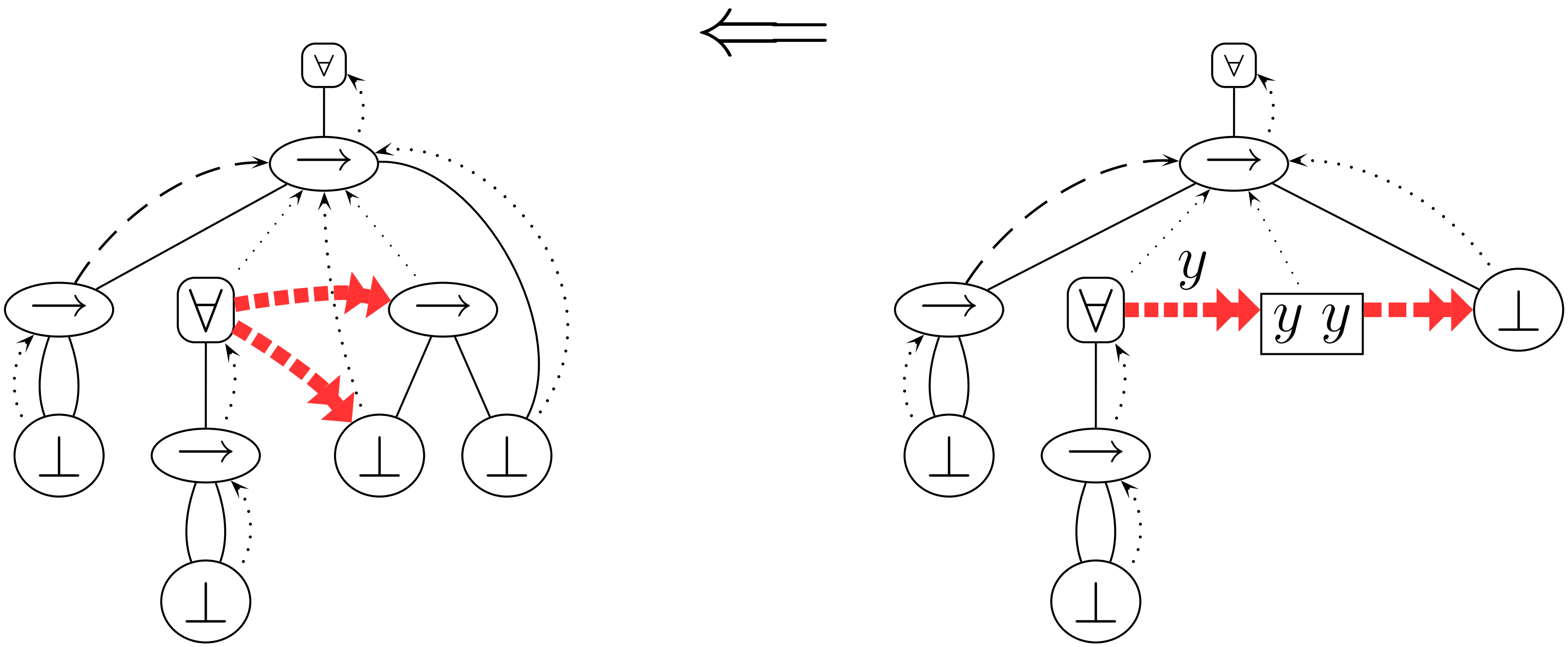
(4) existential elimination

$$\Longrightarrow$$

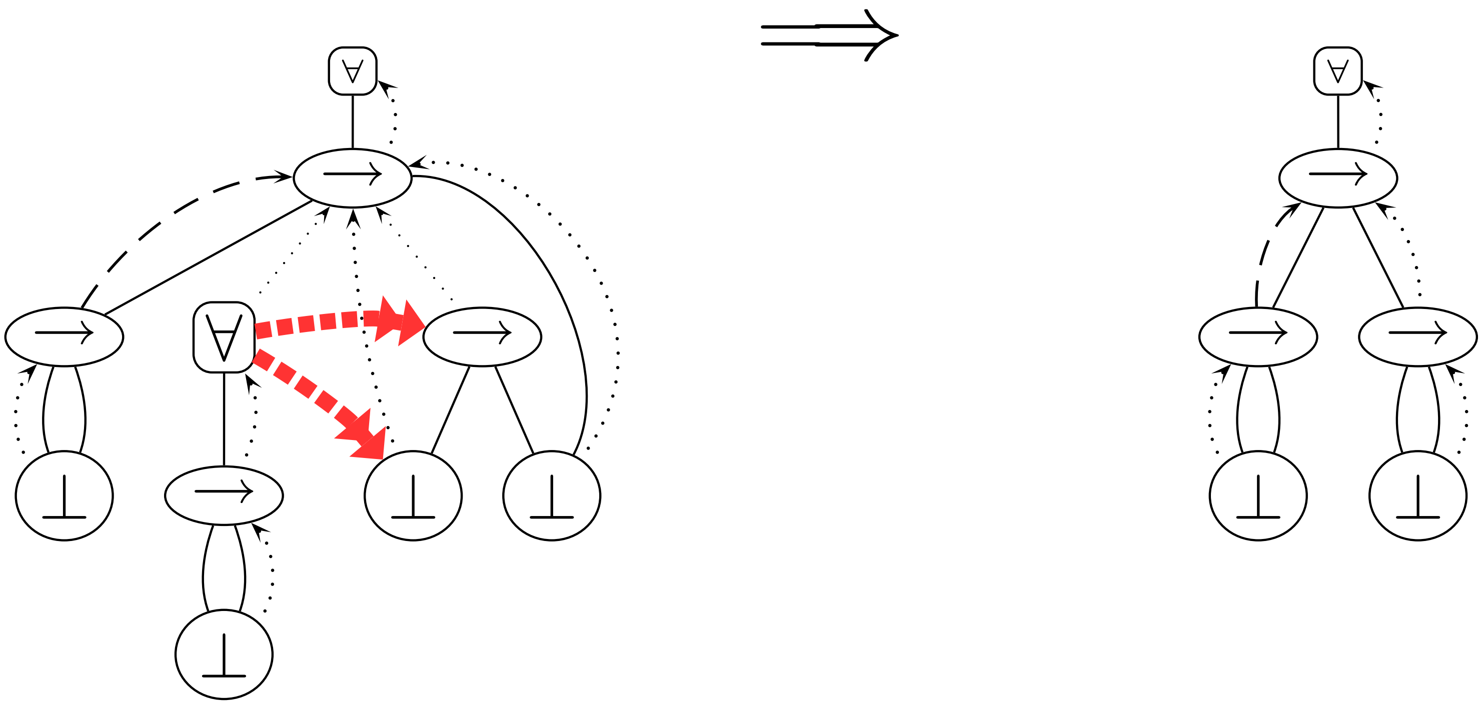




(5)



(6) by definition



(7) many steps

The End