Structural Refinement Types

TyDe '22, David Binder, Ingo Skupin, David Läwen, Klaus Ostermann

$$\textbf{def}\ \texttt{pred} \coloneqq \lambda x.\textbf{case}\ x\ \textbf{of}\ \{S(n) \Rightarrow n\}$$

$$\textbf{def}\ \texttt{pred} \coloneqq \lambda x.\textbf{case}\ x\ \textbf{of}\ \{S(n) \Rightarrow n\}$$

$$\mathbb{N} \to \mathbb{N}$$

$$\textbf{def}\ \texttt{pred} := \lambda x.\textbf{case}\ x\ \textbf{of}\ \{S(n) \Rightarrow n\}$$

$$\mathbb{N} \to \mathbb{N}$$

Consider: pred Z

$$\textbf{def}\ \texttt{pred} \coloneqq \lambda x.\textbf{case}\ x\ \textbf{of}\ \{S(n) \Rightarrow n\}$$

$$\forall \alpha\,.\,\langle\ S(\alpha)\ \rangle \to \alpha$$

$$\textbf{def}\ \texttt{pred} := \lambda x.\textbf{case}\ x\ \textbf{of}\ \{S(n) \Rightarrow n\}$$

$$\forall \alpha\, .\, \langle\ S(\alpha)\ \rangle \to \alpha$$

Consider: pred S(true)

$$\textbf{def}\ \texttt{pred} \coloneqq \lambda x.\textbf{case}\ x\ \textbf{of}\ \{S(n) \Rightarrow n\}$$

$$\langle \mathbb{N} \mid S(\mathbb{N}^{\top}) \rangle \to \mathbb{N}^{\top}$$

$$\textbf{def}\ \texttt{pred} := \lambda x.\textbf{case}\ x\ \textbf{of}\ \{S(n) \Rightarrow n\}$$

$$\langle \mathbb{N} \,|\, S(\mathbb{N}^{\top}) \rangle \to \mathbb{N}^{\top}$$

$$\mathbb{N}^{\top} := \mu\alpha\,.\,\langle \mathbb{N} \,|\, Z, S(\alpha) \rangle$$

Our simple idea:

Nominal

$$\mathbb{N} \to \mathbb{N}$$

Structural / Polymorphic Variant

$$\forall \alpha \,.\, \langle\ S(\alpha)\ \rangle \to \alpha$$

$$\langle \mathbb{N} \,|\, S(\mu\alpha \,.\, \langle \mathbb{N} \,|\, Z, S(\alpha) \rangle) \rangle \to \mu\alpha \,.\, \langle \mathbb{N} \,|\, Z, S(\alpha) \rangle$$

Structural Refinement Type

Our simple idea:

Nominal

$\mathbb{N} \to \mathbb{N}$

Structural / Polymorphic Variant

$\forall \alpha \,.\, \langle\; S(\alpha) \;\rangle \to \alpha$

$\langle \mathbb{N} \mid S(\mu\alpha \,.\, \langle \mathbb{N} \mid Z, S(\alpha) \rangle) \rangle \to \mu\alpha \,.\, \langle \mathbb{N} \mid Z, S(\alpha) \rangle$

Structural Refinement Type

Our simple idea:

Nominal

$$\mathbb{N} \to \mathbb{N}$$

Structural / Polymorphic Variant

$$\forall \alpha \,.\, \langle\; S(\alpha) \;\rangle \to \alpha$$

$$\langle \mathbb{N} \mid S(\mu\alpha \,.\, \langle \mathbb{N} \mid Z, S(\alpha) \rangle) \rangle \to \mu\alpha \,.\, \langle \mathbb{N} \mid Z, S(\alpha) \rangle$$

Structural Refinement Type

Thankfully, we didn't have to do any of the hard work!

Subtyping

Subtyping

- Refinement types require subtyping: Any function accepting natural numbers should also accept non-zero natural numbers.

Subtyping

- Refinement types require subtyping: Any function accepting natural numbers should also accept non-zero natural numbers.
- The combination of subtyping, parametric polymorphism and complete inference of principal types is hard.

Subtyping

- Refinement types require subtyping: Any function accepting natural numbers should also accept non-zero natural numbers.
- The combination of subtyping, parametric polymorphism and complete inference of principal types is hard.
- We build upon the work on **algebraic subtyping (AS)** of Dolan (2017), Dolan and Mycroft (2017), and Parreaux (2020) who showed how to do this.

Subtyping

- Refinement types require subtyping: Any function accepting natural numbers should also accept non-zero natural numbers.
- The combination of subtyping, parametric polymorphism and complete inference of principal types is hard.
- We build upon the work on **algebraic subtyping (AS)** of Dolan (2017), Dolan and Mycroft (2017), and Parreaux (2020) who showed how to do this.
- Very similar idea to Hindley-Milner (HM) type-inference, but instead of type equality constraints $\sigma \sim \tau$ we generate type inequality constraints $\sigma <: \tau$.

Typing rules for constructors

$$\frac{\Gamma \vdash e : \mathbb{N}}{\Gamma \vdash S(e) : \mathbb{N}}\ \mathrm{S}_{Nominal}$$

$$\frac{\Gamma \vdash e : \sigma}{\Gamma \vdash \text{`}\mathsf{S}(e) : \langle\, \text{`}\mathsf{S}(\sigma)\,\rangle}\ \mathrm{S}_{Structural}$$

$$\frac{\Gamma \vdash e : \sigma \qquad \sigma <: \mu\alpha.\langle\, \mathbb{N} \mid Z, S(\alpha)\,\rangle}{\Gamma \vdash S(e) : \langle\, \mathbb{N} \mid S(\sigma)\,\rangle}\ \mathrm{S}_{Refinement}$$

Typing rules for constructors

$$\frac{\Gamma \vdash e : \mathbb{N}}{\Gamma \vdash S(e) : \mathbb{N}}\ \mathrm{S}_{Nominal}$$

$$\frac{\Gamma \vdash e : \sigma}{\Gamma \vdash \texttt{`S}(e) : \langle\ \texttt{`S}(\sigma)\ \rangle}\ \mathrm{S}_{Structural}$$

$$\frac{\Gamma \vdash e : \sigma \qquad \sigma <: \mu\alpha.\langle\ \mathbb{N} \mid Z, S(\alpha)\ \rangle}{\Gamma \vdash S(e) : \langle\ \mathbb{N} \mid S(\sigma)\ \rangle}\ \mathrm{S}_{Refinement}$$

Typing rules for constructors

$$\frac{\Gamma \vdash e : \mathbb{N}}{\Gamma \vdash S(e) : \mathbb{N}}\ \mathrm{S}_{Nominal}$$

$$\frac{\Gamma \vdash e : \sigma}{\Gamma \vdash \grave{}\mathsf{S}(e) : \langle\ \grave{}\mathsf{S}(\sigma)\ \rangle}\ \mathrm{S}_{Structural}$$

$$\frac{\Gamma \vdash e : \sigma \qquad \sigma <: \mu\alpha.\langle\ \mathbb{N} \mid Z, S(\alpha)\ \rangle}{\Gamma \vdash S(e) : \langle\ \mathbb{N} \mid S(\sigma)\ \rangle}\ \mathrm{S}_{Refinement}$$

Typing rules for pattern matches

$$\frac{\Gamma \vdash e : \mathbb{N} \qquad \Gamma \vdash e_Z : \tau \qquad \Gamma, x : \mathbb{N} \vdash e_S : \tau}{\Gamma \vdash \textbf{case}\ e\ \textbf{of}\ \{Z \Rightarrow e_Z, S(x) \Rightarrow e_S\} : \tau}\ \textsc{Case}^{\mathbb{N}}_{\mathit{Nominal}}$$

$$\frac{\Gamma \vdash e : \langle\ \text{`}\mathsf{S}(\tau)\ \rangle \qquad \Gamma, x : \tau \vdash e_S : \rho}{\Gamma \vdash \textbf{case}\ e\ \textbf{of}\ \{\text{`}\mathsf{S}(x) \Rightarrow e_S\} : \rho}\ \textsc{Case}^{S}_{\mathit{Structural}}$$

$$\frac{\begin{array}{c}\langle\ \mathbb{N} \mid \emptyset\ \rangle <: \tau <: \mu\alpha.\langle\ \mathbb{N} \mid Z, S(\alpha)\ \rangle \\ \Gamma \vdash e : \langle\ \mathbb{N} \mid S(\tau)\ \rangle \\ \Gamma, x : \tau \vdash e_S : \rho\end{array}}{\Gamma \vdash \textbf{case}\ e\ \textbf{of}\ \{S(x) \Rightarrow e_S\} : \rho}\ \textsc{Case}^{S}_{\mathit{Refinement}}$$

Typing rules for pattern matches

$$\frac{\Gamma \vdash e : \mathbb{N} \qquad \Gamma \vdash e_Z : \tau \qquad \Gamma, x : \mathbb{N} \vdash e_S : \tau}{\Gamma \vdash \textbf{case}\ e\ \textbf{of}\ \{Z \Rightarrow e_Z, S(x) \Rightarrow e_S\} : \tau}\ \textsc{Case}^{\mathbb{N}}_{\mathit{Nominal}}$$

$$\frac{\Gamma \vdash e : \langle\ \text{`S}(\tau)\ \rangle \qquad \Gamma, x : \tau \vdash e_S : \rho}{\Gamma \vdash \textbf{case}\ e\ \textbf{of}\ \{\text{`S}(x) \Rightarrow e_S\} : \rho}\ \textsc{Case}^{S}_{\mathit{Structural}}$$

$$\frac{\begin{array}{c}\langle\ \mathbb{N} \mid \emptyset\ \rangle <: \tau <: \mu\alpha.\langle\ \mathbb{N} \mid Z, S(\alpha)\ \rangle \\ \Gamma \vdash e : \langle\ \mathbb{N} \mid S(\tau)\ \rangle \\ \Gamma, x : \tau \vdash e_S : \rho\end{array}}{\Gamma \vdash \textbf{case}\ e\ \textbf{of}\ \{S(x) \Rightarrow e_S\} : \rho}\ \textsc{Case}^{S}_{\mathit{Refinement}}$$

Typing rules for pattern matches

$$\frac{\Gamma \vdash e : \mathbb{N} \qquad \Gamma \vdash e_Z : \tau \qquad \Gamma, x : \mathbb{N} \vdash e_S : \tau}{\Gamma \vdash \textbf{case}\ e\ \textbf{of}\ \{Z \Rightarrow e_Z, S(x) \Rightarrow e_S\} : \tau}\ \textsc{Case}^{\mathbb{N}}_{Nominal}$$

$$\frac{\Gamma \vdash e : \langle\, \text{`S}(\tau)\, \rangle \qquad \Gamma, x : \tau \vdash e_S : \rho}{\Gamma \vdash \textbf{case}\ e\ \textbf{of}\ \{\text{`S}(x) \Rightarrow e_S\} : \rho}\ \textsc{Case}^{S}_{Structural}$$

$$\frac{\begin{array}{c}\langle\, \mathbb{N} \mid \emptyset\, \rangle <: \tau <: \mu\alpha.\langle\, \mathbb{N} \mid Z, S(\alpha)\, \rangle \\ \Gamma \vdash e : \langle\, \mathbb{N} \mid S(\tau)\, \rangle \\ \Gamma, x : \tau \vdash e_S : \rho\end{array}}{\Gamma \vdash \textbf{case}\ e\ \textbf{of}\ \{S(x) \Rightarrow e_S\} : \rho}\ \textsc{Case}^{S}_{Refinement}$$

What will you find in the paper?

Parameterized Types

How should we refine parameterized type

[true, "hello tyde"]

Parameterized Types

How should we refine parameterized type

$[\texttt{true}, \text{"hello tyde"}]$

$[\texttt{true}, \text{"hello tyde"}] : [\mathbb{B}, \texttt{String}]$

Parameterized Types

How should we refine parameterized type

$[\texttt{true}, \text{"hello tyde"}]$

$[\texttt{true}, \text{"hello tyde"}] : [\mathbb{B}, \texttt{String}]$

$[\texttt{true}, \text{"hello tyde"}] : [\_, \_] @ (\mathbb{B} \vee \texttt{String})$

Parameterized Types

How should we refine parameterized type

[$\mathtt{true}$, "hello tyde"]

[$\mathtt{true}$, "hello tyde"] : [$\mathbb{B}$, $\mathtt{String}$]

[$\mathtt{true}$, "hello tyde"] : [_, _] @ ($\mathbb{B} \vee \mathtt{String}$)

Parameterized Types

How should we refine parameterized type

$[\texttt{true}, \text{"hello tyde"}]$

$[\texttt{true}, \text{"hello tyde"}] : [\mathbb{B}, \texttt{String}]$

$[\texttt{true}, \text{"hello tyde"}] : [\_, \_] @ (\mathbb{B} \vee \texttt{String})$

Parameterized Types

How should we refine parameterized type

$$[\texttt{true}, \text{"hello tyde"}]$$

$$[\texttt{true}, \text{"hello tyde"}] : [\mathbb{B}, \texttt{String}]$$

$$[\texttt{true}, \text{"hello tyde"}] : [\_, \_] @ (\mathbb{B} \vee \texttt{String})$$

Technical details

Technical details

Constraint generation: $\Gamma \vdash e : \tau \rightsquigarrow \Xi$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau \rightsquigarrow \emptyset}\ \textsc{G-Var} \qquad \frac{\Gamma, x : \beta^? \vdash e : \tau \rightsquigarrow \Xi \qquad \mathit{Fresh}(\beta^?)}{\Gamma \vdash \lambda x.e : \beta^? \to \tau \rightsquigarrow \Xi}\ \textsc{G-Lam}$$

$$\frac{\Gamma \vdash e_1 : \sigma_1 \rightsquigarrow \Xi_1 \qquad \Gamma \vdash e_2 : \sigma_2 \rightsquigarrow \Xi_2 \qquad \mathit{Fresh}(\beta^?)}{\Gamma \vdash e_1\ e_2 : \beta^? \rightsquigarrow \{\sigma_1 <: \sigma_2 \to \beta^?\} \cup \Xi_1 \cup \Xi_2}\ \textsc{G-App}$$

$$\frac{\overline{\Gamma \vdash e : \tau \rightsquigarrow \Xi} \qquad \forall \overline{\alpha}, \overline{\alpha'}. C(\overline{\sigma}) : N(\overline{\alpha}; \overline{\alpha'}) \in \mathit{Ctors} \qquad \mathit{Fresh}(\overline{\beta^?}, \overline{\beta'^?})}{\Gamma \vdash C(\overline{e}) : \langle\, N(\overline{\alpha}; \overline{\alpha'}) \mid C(\overline{\sigma})\,\rangle @(\overline{\beta^?}; \overline{\beta'^?}) \rightsquigarrow \left\{\overline{\tau <: [\![\sigma]\!]^{\top}_{N}[\overline{\beta^?/\alpha}, \overline{\beta'^?/\alpha'}]}\right\} \cup (\bigcup_i \Xi_i)}\ \textsc{G-Ctor}$$

$$\frac{\begin{array}{c}\overline{\Gamma, \overline{x : \beta^?} \vdash e : \tau \rightsquigarrow \Xi} \\ \Gamma \vdash e : \sigma \rightsquigarrow \Xi\end{array} \qquad \mathit{Fresh}(\overline{\beta^?}, \gamma^?, \overline{\delta^?}, \overline{\delta'^?}) \qquad \begin{array}{c}\overline{C(\overline{\beta^?}) \heartsuit_N C(\overline{\sigma}[\overline{\delta^?/\alpha}, \overline{\delta'^?/\alpha'}]) \rightsquigarrow \Xi_\heartsuit} \\ \overline{\forall \overline{\alpha}, \overline{\alpha'}. C(\overline{\sigma}) : N(\overline{\alpha}; \overline{\alpha'})} \subseteq \mathit{Ctors}\end{array}}{\Gamma \vdash \mathbf{case}\ e\ \mathbf{of}\ \{\overline{C(\overline{x}) \Rightarrow e}\} : \gamma^? \rightsquigarrow \left\{\sigma <: \langle\, N(\overline{\alpha}; \overline{\alpha'}) \mid \overline{C(\overline{\sigma})}\,\rangle @(\overline{\delta^?}; \overline{\delta'^?}),\ \overline{\tau <: \gamma^?},\right\} \cup \Xi \cup \{\bigcup_i \Xi_i\} \cup \{\bigcup_i \Xi_{\heartsuit,i}\}}\ \textsc{G-Case}$$

$$\frac{}{C(\overline{\beta^?}) \heartsuit_N C(\overline{\tau}) \rightsquigarrow \left\{\overline{[\![\tau]\!]^{\bot}_{N} <: \beta^? <: [\![\tau]\!]^{\top}_{N}}\right\}}\ \textsc{G-Compat}$$

(a) Constraint generation rules. Inputs are contexts and terms, outputs are types and constraint sets.

Technical details

Constraint solver step: $S \twoheadrightarrow S'$

$$\frac{q \in ca}{ca; q, qs \vdash bs \twoheadrightarrow ca; qs \vdash bs}\ \textsc{CacheHit}$$

$$\frac{q \notin ca \quad q = \alpha^? <: \sigma \quad bs(\alpha^?) = lbs <: \alpha^? <: ubs}{ca; q, qs \vdash bs \twoheadrightarrow ca, q; \{lb <: \sigma\}_{lb \in lbs}, qs \vdash bs[\alpha^? \mapsto lbs <: \alpha^? <: \{ubs, \sigma\}]}\ \textsc{UpperBound}$$

$$\frac{q \notin ca \quad q = \sigma <: \alpha^? \quad \sigma \notin \textsc{TyVar} \quad bs(\alpha^?) = lbs <: \alpha^? <: ubs}{ca; q, qs \vdash bs \twoheadrightarrow ca, q; \{\sigma <: ub\}_{ub \in ubs}, qs \vdash bs[\alpha^? \mapsto \{\sigma, lbs\} <: \alpha^? <: ubs]}\ \textsc{LowerBound}$$

$$\frac{q \notin ca \quad q = \sigma_1 <: \sigma_2 \quad \sigma_1 \notin \textsc{TyVar} \quad \sigma_2 \notin \textsc{TyVar} \quad \mathbf{Sub}(q) = qs'}{ca; q, qs \vdash bs \twoheadrightarrow ca, q; qs' +\!\!+\, qs \vdash bs}\ \textsc{SubOk}$$

$$\frac{q \notin ca \quad q = \sigma_1 <: \sigma_2 \quad \sigma_1 \notin \textsc{TyVar} \quad \sigma_2 \notin \textsc{TyVar} \quad \mathbf{Sub}(q) = \mathbf{Fail}}{ca; q, qs \vdash bs \twoheadrightarrow \mathbf{Fail}}\ \textsc{SubFail}$$

(b) The biunification algorithm.

Decomposing non-atomic constraints: $\mathbf{Sub}(-) : q \to \overline{q}/\mathbf{Fail}$

$$\begin{array}{lcl}
\mathbf{Sub}(\tau <: \top) & := & \emptyset \\
\mathbf{Sub}(\bot <: \sigma) & := & \emptyset \\
\mathbf{Sub}(\tau_1 \sqcup \tau_2 <: \sigma) & := & \{\tau_1 <: \sigma, \tau_2 <: \sigma\} \\
\mathbf{Sub}(\tau <: \sigma_1 \sqcap \sigma_2) & := & \{\tau <: \sigma_1, \tau <: \sigma_2\} \\
\mathbf{Sub}(\tau <: \mu\alpha.\sigma) & := & \{\tau <: \sigma[\mu\alpha.\sigma/\alpha]\} \\
\mathbf{Sub}(\mu\alpha.\tau <: \sigma) & := & \{\tau[\mu\alpha.\tau/\alpha] <: \sigma\} \\
\mathbf{Sub}(\alpha <: \alpha) & := & \emptyset \\
\mathbf{Sub}(\sigma_1 \to \tau_1 <: \sigma_2 \to \tau_2) & := & \{\sigma_2 <: \sigma_1, \tau_1 <: \tau_2\} \\
\mathbf{Sub}(\tau @ (\overline{\sigma}; \overline{\rho}) <: \tau' @ (\overline{\sigma'}; \overline{\rho'})) & := & \{\tau <: \tau', \overline{\sigma <: \sigma'}, \overline{\rho' <: \rho}\} \\
\mathbf{Sub}(\langle\, N(\overline{\alpha}; \overline{\alpha'}) \mid \emptyset \,\rangle <: \langle\, N(\overline{\beta}; \overline{\beta'}) \mid \psi \,\rangle) & := & \emptyset \\
\mathbf{Sub}(\langle\, N(\overline{\alpha}; \overline{\alpha'}) \mid C(\overline{\tau}), \varphi \,\rangle <: \langle\, N(\overline{\beta}; \overline{\beta'}) \mid C(\overline{\sigma}), \psi \,\rangle) & := & \left\{\overline{\tau_i <: \sigma_i[\overline{\alpha}/\overline{\beta}]}\right\} \cup \mathbf{Sub}(\langle\, N(\overline{\alpha}; \overline{\alpha'}) \mid \varphi \,\rangle <: \langle\, N(\overline{\beta}; \overline{\beta'}) \mid \psi \,\rangle)
\end{array}$$

(c) Decomposing subtyping constraints.

Constraint generation: $\Gamma \vdash e : \tau \rightsquigarrow \Xi$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau \rightsquigarrow \emptyset}\ \text{G-Var}$$

$$\frac{\Gamma, x : \beta^? \vdash e : \tau \rightsquigarrow \Xi \quad Fresh(\beta^?)}{\Gamma \vdash \lambda x.e : \beta^? \to \tau \rightsquigarrow \Xi}\ \text{G-Lam}$$

$$\frac{\Gamma \vdash e_1 : \sigma_1 \rightsquigarrow \Xi_1 \quad \Gamma \vdash e_2 : \sigma_2 \rightsquigarrow \Xi_2 \quad Fresh(\beta^?)}{\Gamma \vdash e_1\ e_2 : \beta^? \rightsquigarrow \{\sigma_1 <: \sigma_2 \to \beta^?\} \cup \Xi_1 \cup \Xi_2}\ \text{G-App}$$

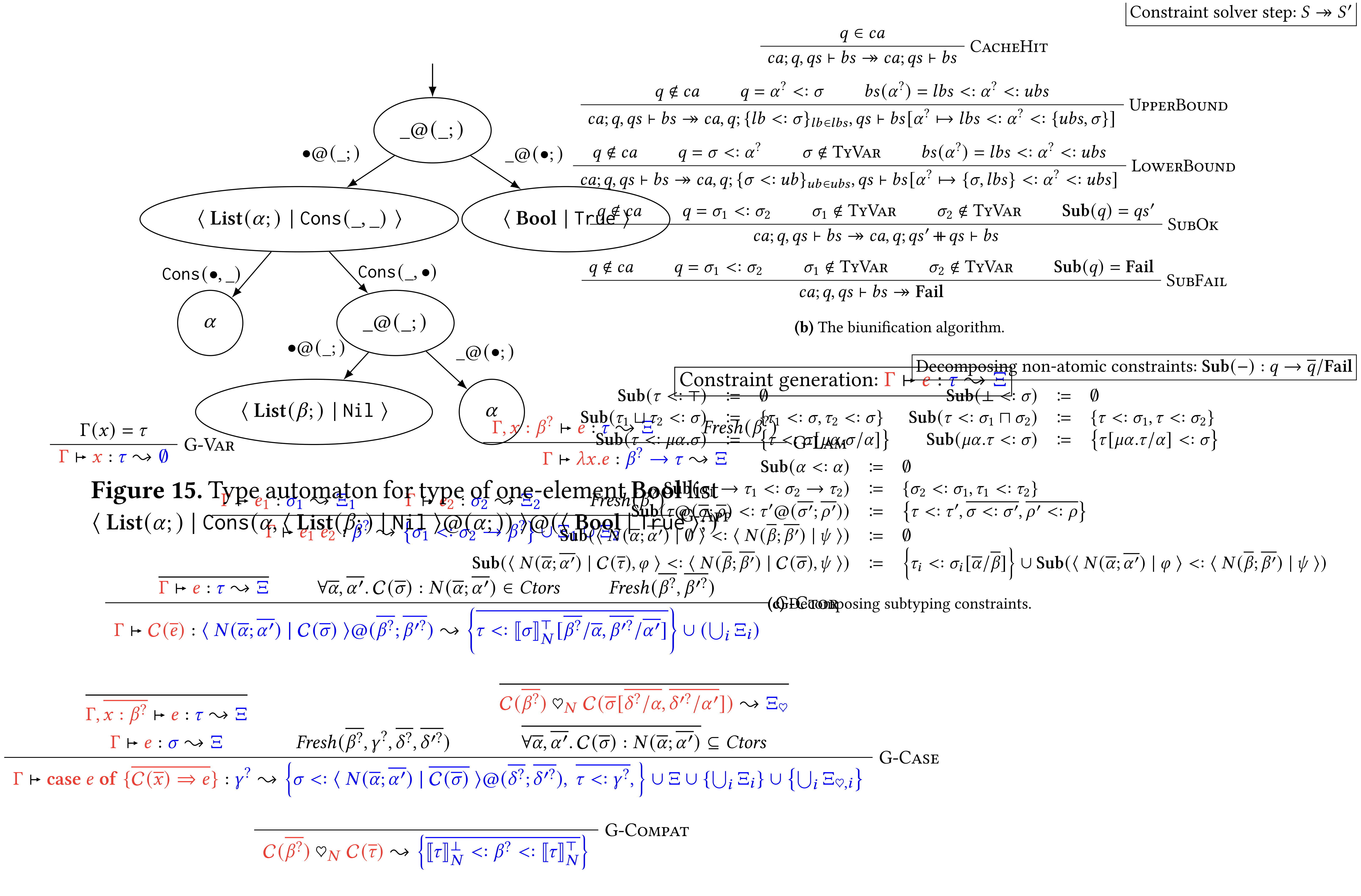
$$\frac{\overline{\Gamma \vdash e : \tau \rightsquigarrow \Xi} \quad \forall \overline{\alpha}, \overline{\alpha'}.\, C(\overline{\sigma}) : N(\overline{\alpha}; \overline{\alpha'}) \in Ctors \quad Fresh(\overline{\beta^?}, \overline{\beta'^?})}{\Gamma \vdash C(\overline{e}) : \langle\, N(\overline{\alpha}; \overline{\alpha'}) \mid C(\overline{\sigma}) \,\rangle @ (\overline{\beta^?}; \overline{\beta'^?}) \rightsquigarrow \left\{\overline{\tau <: [\![\sigma]\!]^{\top}_{N}[\overline{\beta^?}/\overline{\alpha}, \overline{\beta'^?}/\overline{\alpha'}]}\right\} \cup (\bigcup_i \Xi_i)}\ \text{G-Ctor}$$

$$\frac{\overline{\Gamma, \overline{x : \beta^?} \vdash e : \tau \rightsquigarrow \Xi} \quad \Gamma \vdash e : \sigma \rightsquigarrow \Xi \quad Fresh(\overline{\beta^?}, \gamma^?, \overline{\delta^?}, \overline{\delta'^?}) \quad \overline{C(\overline{\beta^?}) \heartsuit_N C(\overline{\sigma}[\overline{\delta^?}/\overline{\alpha}, \overline{\delta'^?}/\overline{\alpha'}]) \rightsquigarrow \Xi_{\heartsuit}} \quad \overline{\forall \overline{\alpha}, \overline{\alpha'}.\, C(\overline{\sigma}) : N(\overline{\alpha}; \overline{\alpha'})} \subseteq Ctors}{\Gamma \vdash \mathbf{case}\ e\ \mathbf{of}\ \{\overline{C(\overline{x}) \Rightarrow e}\} : \gamma^? \rightsquigarrow \left\{\sigma <: \langle\, N(\overline{\alpha}; \overline{\alpha'}) \mid \overline{C(\overline{\sigma})} \,\rangle @ (\overline{\delta^?}; \overline{\delta'^?}),\ \overline{\tau <: \gamma^?},\right\} \cup \Xi \cup \{\bigcup_i \Xi_i\} \cup \{\bigcup_i \Xi_{\heartsuit,i}\}}\ \text{G-Case}$$

$$\frac{}{C(\overline{\beta^?}) \heartsuit_N C(\overline{\tau}) \rightsquigarrow \left\{\overline{[\![\tau]\!]^{\bot}_{N} <: \beta^? <: [\![\tau]\!]^{\top}_{N}}\right\}}\ \text{G-Compat}$$

(a) Constraint generation rules. Inputs are contexts and terms, outputs are types and constraint sets.

Technical details



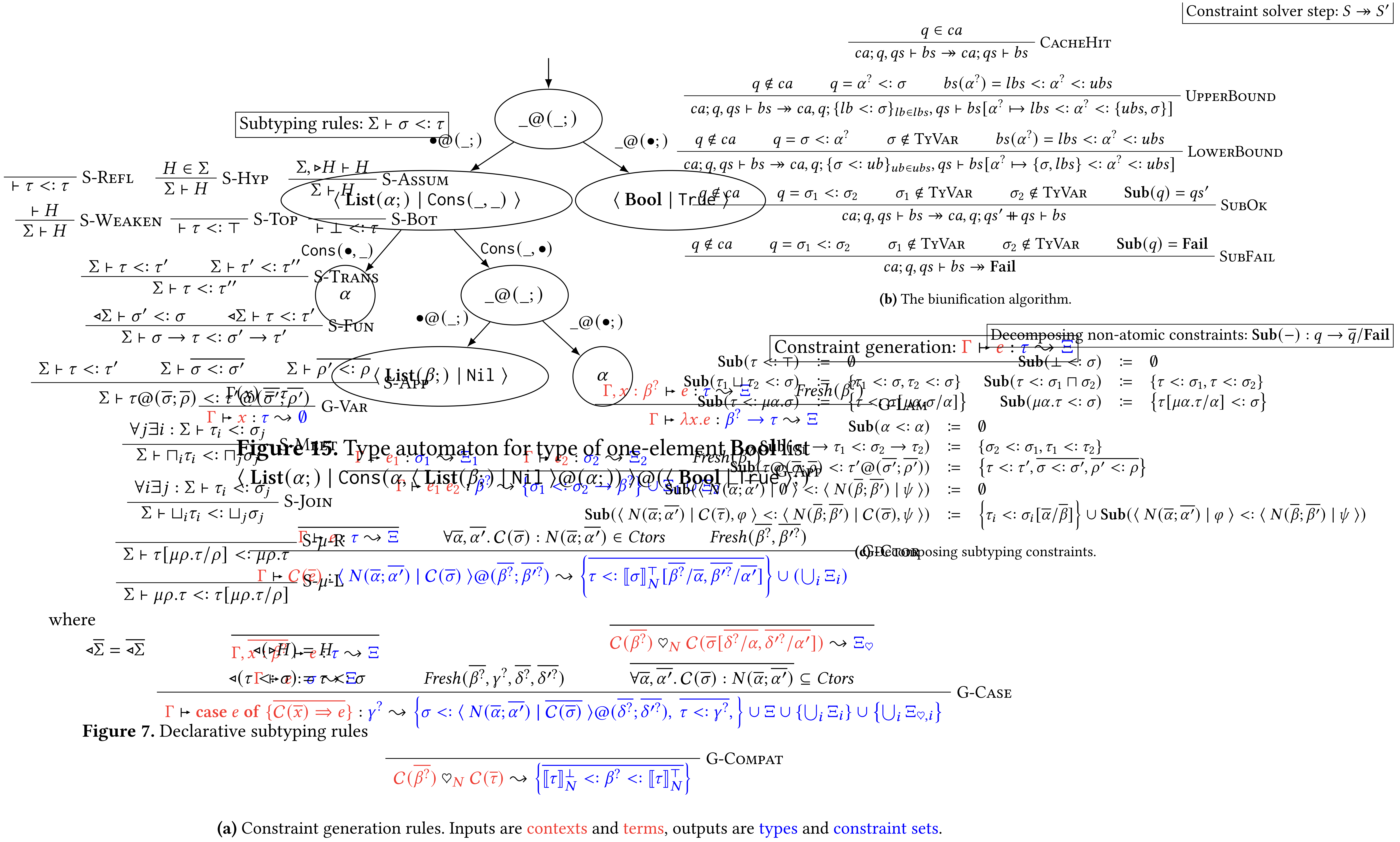
(b) The biunification algorithm.

Figure 15. Type automaton for type of one-element Bool list

(c) Decomposing subtyping constraints.

(a) Constraint generation rules. Inputs are contexts and terms, outputs are types and constraint sets.

Technical details



Takeaways

Takeaways

- Easy to implement if you already use algebraic subtyping.

Takeaways

- Easy to implement if you already use algebraic subtyping.
- Expressive enough for many interesting use cases.

Takeaways

- Easy to implement if you already use algebraic subtyping.
- Expressive enough for many interesting use cases.
- Does not require anything but familiar type inference machinery.

What remains to be done?

Future work

Future work

- Develop formal metatheory.

Future work

- Develop formal metatheory.
- Investigate expressivity.

Future work

- Develop formal metatheory.
- Investigate expressivity.
- Larger case studies.

Future work

- Develop formal metatheory.
- Investigate expressivity.
- Larger case studies.
- Usability engineering.

That was my presentation.

What do **you** want to know?