Invariant Inference via GADTs

BY ŁUKASZ STAFINIAK

Institute of Computer Science, University of Wrocław

Abstract

GADTs can express data invariants and program correctness in functional languages. Previous approaches to type inference and type checking for GADTs focus on efficient decidability and predictability. We present a framework and algorithms for a programmer assistant that automatically generates types and invariants based on the GADTs system HMG(X) but without mandatory type annotations, with some completeness guarantees. It is inspired by prior work on polymorphic recursion, type inference via constraint abduction, and automatic generation of loop invariants. We extend the inference to discover existentially quantified descriptions of results of functions (i.e. postconditions, while the remaining part of function invariants are preconditions).

Keywords: invariant inference, type inference, GADTs, constraint abduction

1 Introduction to Generalized Algebraic Data Types

Type systems are established natural deduction-style means to reason about programs. Dependent types can represent arbitrarily complex properties as they use the same language for both types and programs (the type of value returned by a function can itself be a function of the argument). GADTs bring some of that expressivity to type systems that deal with data-types, but with limitations: the semantic of types becomes simpler than that of programs (for example, deciding type equality can be very simple). GADTs introduce the ability of reasoning about return type by case analysis of the input value, while keeping the benefits of separate semantics. Our type system for GADTs differs from others in that we do not require any type (or invariant) annotations, even on recursive functions.

Consider a function `eval` defined by cases over a datatype Term(α) with constructors `Lit`: Int $\to$ Term(Int), `IsZero`: Term(Int) $\to$ Term(Bool), `If`: $\forall\alpha$.Term(Bool) $\to$ Term(α) $\to$ Term(α) $\to$ Term(α). In a type system with Generalized Algebraic Data Types, the result of reduction of the typing problem for `eval` to constraint solving resembles the following constraint problem (side by side with the corresponding parts of the program): table 1 – from which we would like to find the solution $\alpha = \text{Term}(\beta)$, $\tau = \text{Term}(\beta) \to \beta$, leading to the inferred type `eval`: $\forall\beta.\text{Term}(\beta) \to \beta$. For more information about constraint-based type inference for GADTs, consult [12].

$\exists\tau, \alpha, \beta \forall\gamma.\tau \doteq \alpha \to \beta \ \wedge$	`eval x = case x of`
$(\alpha \doteq \text{Term(Int)} \Rightarrow \beta \doteq \text{Int}) \ \wedge$	`  Lit y -> y`
$(\alpha \doteq \text{Term(Bool)} \Rightarrow \beta \doteq \text{Bool}) \ \wedge$	`  IsZero y -> (eval y)=0`
$(\alpha \doteq \text{Term}(\gamma) \Rightarrow \beta \doteq \gamma)$	`  If y1 y2 y3 ->`
	`    if eval y1 then eval y2`
	`    else eval y3`

Table 1. Simplified constraint for typing `eval`

Now we face an example the type system with GADTs needs further extension to handle. We will collect elements of a list with length that meet a given predicate. We expect the system to know that the resulting list will be no longer than the input list. In OCaml syntax:

```
type ('a, _) list =
  | LNil : ('a, 0) list
  | LCons : ('a, 'n) list -> ('a, 'n+1) list


let rec filter f =
  function LNil -> LNil
    | LCons (x, l) when f x ->
       let l' = filter l in LCons (x, l')
    | LCons (_, l) ->
       let l' = filter l in l'
```

The real type of the `filter` function is:

$$\forall\alpha n.(\alpha \to \text{bool}) \to \text{List}(\alpha, n) \to \exists k[k \leqslant n].\text{List}(\alpha, k)$$

This type motivates extensions to the type system, developed in this work, which allows for existential quantification over return values of pattern matching.

1.1 Related work

In the tradition of the Milner-Mycroft type system (see [2]), we modify the HMG(X) type system from [12] by

dropping the type specifications on recursive definitions from program terms. We also naturally restrict it by limiting the user-specified and inferred invariant constraints to use conjunction as the only logical connective. As these specifications express recursion invariants, we use the traditional framework for invariant generation of [1]. We feel that the application to recursive definitions in functional languages of the technique traditionally applied to programs with loops in imperative languages is under-used. Initially we were only aware of the work [3], which applies Dijkstra's weakest precondition calculus to refinement types. A work similar to ours could be done by application of the weakest precondition calculus to the Hoare logic of [10], with the conditions inserted by type inference.

The work in [14], although it is advertised as focused on dependent types, can be seen as extending [3] with reasoning by Boolean cases. Their programming language and type system is in several ways less expressive than the ML language with polymorphic recursion and the full GADTs type system: no inductive types (and therefore no pattern matching), refinement predicates over integers only instead of over arbitrary domains including types. Still, the inclusion of reasoning by cases and development of methods to actually find the refinement predicates, make [14] closer to our results.

Our algorithm eliminates implications in a way similar to [13], but using a slightly different definition of abduction. Use of abduction in [13] is related to the work in [6] and [7], where a more complete abduction algorithm is provided. Our algorithm is extensible to any constraint domain, by providing an abduction algorithm and a quantified conjunctive constraints solution algorithm. It necessarily includes the domain of equations over (free) algebraic terms. The full exposition of our use of abduction is in [15] and its accompanying technical report [16].

There is a surge of recent work on type inference for GADTs, not contributing to our approach. Works such as [9] (older), [11], [5] and [4] modify the GADTs type system to make it more amenable to type inference (rejecting some reasonable programs as untypable), and develop less declarative inference algorithms. These works also do not allow other domains (than the free term algebra) to express invariants. [4] stands out from our point of view as it handles type inference for polymorphic recursion (in the same way as we do, by iteration).

Due to space constraints and technical character of proofs for presented propositions, the proofs are delegated to accompanying technical reports [17] and [18].

1.2 Notation

By the bar $\bar{e}$ we denote a sequence (or a set, depending on context) of elements e, by $\#$ we denote disjointness. With a free index i, $\bar{e_i}$ denotes $(e_1, ..., e_n)$ for some n associated with the index i; similarly, $\wedge_i \Phi_i$ denotes $\Phi_1 \wedge ... \wedge \Phi_n$. For convenience, we identify syntactically a conjunction of atoms $\wedge_i c_i$ with a set of atoms $\{c_1, ..., c_n\}$.

In some contexts, for a quantifier prefix $\mathcal{Q}$ we write $\mathcal{Q}$ to denote the set of variables quantified by $\mathcal{Q}$. Let FV be a generic function returning the free variables of any expression. For a set of variables V, let $\exists(V^c).\Phi$ denote $\exists \mathrm{FV}(\Phi) \setminus V.\Phi$, i.e. existential closure of Φ except for variables from V, which are kept free. For a quantifier prefix $\mathcal{Q}$ and variables x, y in $\mathcal{Q}$, by $x <_{\mathcal{Q}} y$ we denote that x is to the left of y in $\mathcal{Q}$ and they are separated by a quantifier alternation, by $x \leqslant_{\mathcal{Q}} y$ that it is not the case that $y <_{\mathcal{Q}} x$. By $\mathcal{Q}[\overline{Q_\alpha \alpha} := \overline{Q_\beta \beta}]$ we denote replacement of Q_α quantification over α with Q_β quantification over β in $\mathcal{Q}$ ($\overline{Q_\alpha}, \overline{Q_\beta} \subset \{\forall, \exists\}$).

Let $T(F)$ be the set of ground terms (i.e. finite trees) for signature F, and $T(F, X)$ the set of (possibly multisorted) terms for signature F and variables X.

By $\Phi[\bar{\alpha} := \bar{t}]$, $\Phi[\overline{\alpha := t}]$, or $\Phi[\alpha_1 := t_1; ...; \alpha_n := t_n]$, we denote a substitution of terms $\bar{t}$ for corresponding variables $\bar{\alpha}$ in the formula Φ (where $\bar{\alpha}$ and $\bar{t}$ are finite sequences of the same length). By $\bar{s} \doteq \bar{t}$ we denote $\wedge_i s_i \doteq t_i$, where $\bar{s} = (s_1, ..., s_n)$ and $\bar{t} = (t_1, ..., t_n)$ for some n. When a substitution has a name, for example $S = [\bar{\alpha} := \bar{t}]$, we write substitution application as $S(\Phi) = \Phi[\bar{\alpha} := \bar{t}]$; we write $\dot{S} = \bar{\alpha} \doteq \bar{t}$; and we denote the substitution S corresponding to a formula $A = \dot{S} = \bar{\alpha} \doteq \bar{t}$ by $\tilde{A}$. We say that a substitution $[\bar{\alpha} := \bar{t}]$ agrees with a quantifier prefix $\mathcal{Q}$, when $\vDash \mathcal{Q}.\bar{\alpha} \doteq \bar{t}$ and in case of $\alpha_1 \doteq \alpha_2 \in \bar{\alpha} \doteq \bar{t}$ for variables α_1, α_2, we have $\alpha_2 \leqslant_{\mathcal{Q}} \alpha_1$.

2 The Language of Constraints

We are interested in a multisorted first-order language with equality $\mathcal{L}$, interpreted in a given model $\mathcal{M}$. The sort of "proper types", denoted s_{ty}, plays a special role. In the current presentation, we will abstract from details of the language, posing the necessary properties as assumptions.

Proposition 1. *If $\vDash D \Rightarrow C$ or $\vDash \forall \bar{\beta}.D \Rightarrow C$, and $\vDash \exists \bar{\alpha}.D$, then $\vDash \exists \bar{\alpha}.C$. More generally, if $\vDash A \Rightarrow \Phi$ and $\vDash \mathcal{Q}.A$, then $\vDash \mathcal{Q}.\Phi$, where $\mathcal{Q}$ is a quantifier prefix.*

We define *solved form formulas* $\mathcal{F}$ to be existentially quantified conjunctions of atoms $\exists \bar{\alpha}.A$. We extend the language and its interpretation in a way reminiscent of existential monadic second order logic, but purely syntactically. It is because we are interested in invariants that are solved form formulas that can share variables with the context of recursive definitions they specify.

Definition 2. *Fix a language $\mathcal{L}$ with a model $\mathcal{M}$. Let ρ be an interpretation of types, that is an assignment of elements of $\mathcal{M}$ to variables in the corresponding sort, extended homomorphically to terms in the standard way. For $\Phi \in \mathcal{L}$, let $\mathcal{M}, \rho \vDash \Phi$ denote the interpretation of a formula Φ in the model $\mathcal{M}$ under the interpretation ρ, in the standard way, for example $\mathcal{M}, \rho \vDash \pi(t)$ if and only if $\pi(\rho(t))$ holds in $\mathcal{M}$, where predicate symbol π in $\mathcal{L}$ corre-*

sponds to predicate π in $\mathcal{M}$, etc.

Extend $\mathcal{L}$ into $\mathcal{L}^{(\delta)}$, resp. $\mathcal{L}^{(\chi)}$, by adding a distinguished variables δ, δ' to sort s_{ty}, resp. a family of unary and binary predicates $\chi_i(\cdot)$ or $\chi_i(\cdot,\cdot)$ over terms of sort s_{ty} (call them predicate variables*). Let $\text{PV}^1(\cdot)$, resp. $\text{PV}^2(\cdot)$ be the set of unary, resp. binary predicate variables in any expression. For a formula Φ in $\mathcal{L}^{(\chi)}$, let $\overline{\chi_i} = \text{PV}^1(\Phi)$, resp. $\overline{\chi_j} = \text{PV}^2(\Phi)$, and let $\overline{\chi_i(\tau_{i,k})}$, resp. $\overline{\chi_j(\tau_{j,k}, \tau'_{j,k})}$ be all occurrences of χ_i, resp. χ_j in Φ. $\mathcal{I}$ is an* interpretation of predicate variables *for Φ when it is an assignment $\mathcal{I} = \overline{\chi_i} := \overline{\exists\bar{\alpha}_i.F_i}\,; \overline{\chi_j} := \overline{\exists\bar{\alpha}_j.F_j}$ such that $\bar{\alpha}_i \# \text{FV}(\wedge_k \tau_{i,k})$ or $\bar{\alpha}_j \# \text{FV}(\wedge_k \tau_{j,k} \ \wedge_k \ \tau'_{j,k})$ and $\overline{\exists\bar{\alpha}_i.F_i}\ \overline{\exists\bar{\alpha}_j.F_j} \subset \mathcal{L}^{(\delta)}$ are solved form formulas. Given $\Phi \in \mathcal{L}^{(\chi)}$, define a statement $\mathcal{M}, \mathcal{I}, \rho \vDash \Phi$ by: $\mathcal{I}$ is an interpretation of predicate variables for Φ, ρ is an interpretation of types, and $\mathcal{M}, \rho \vDash \Phi[\overline{\chi_i(\tau_{i,k})} := \overline{\exists\bar{\alpha}_i.F_i[\delta := \tau_{i,k}]}; \overline{\chi_j(\tau_{j,k}, \tau'_{j,k})} := \overline{\exists\bar{\alpha}_j.F_j[\delta := \tau_{j,k}; \delta' := \tau'_{j,k}]}$, where $\mathcal{I} = \overline{\chi_i} := \overline{\exists\bar{\alpha}_i.F_i}\,; \overline{\chi_j} := \overline{\exists\bar{\alpha}_j.F_j}$.*

Define $\mathcal{M}, \mathcal{I} \vDash \Phi$ as: for all interpretations of types ρ, $\mathcal{M}, \mathcal{I}, \rho \vDash \Phi$. Define $\mathcal{M} \vDash \Phi$ as: for all interpretations of predicate variables $\mathcal{I}$ for Φ, $\mathcal{M}, \mathcal{I} \vDash \Phi$. Often we write $\mathcal{I} \vDash \Phi$, resp. $\vDash \Phi$, instead of $\mathcal{M}, \mathcal{I} \vDash \Phi$, resp. $\mathcal{M} \vDash \Phi$, since the model is fixed. We write $\mathcal{I}, C \vDash \Phi$, resp. $C \vDash \Phi$, for $\mathcal{I} \vDash C \Rightarrow \Phi$, resp. $\vDash C \Rightarrow \Phi$.

We say that a formula Φ is *satisfiable*, if and only if there exists an interpretation of predicate variables $\mathcal{I}$ for Φ, such that $\mathcal{I} \vDash \exists \text{FV}(\Phi).\Phi$. As seen above, we extend the notion of substitution to handle predicate variable atoms, where the replacement of each occurrence of a variable depends on the argument of that variable. For interpretations of predicate variables $\mathcal{I}_1 = \overline{\chi_i^1} := \overline{\exists\bar{\alpha}_i^1.F_i^1}$, $\mathcal{I}_2 = \overline{\chi_i^2} := \overline{\exists\bar{\alpha}_i^2.F_i^2}$ with disjoint domains, we put $\mathcal{I}_1\mathcal{I}_2 = \overline{\chi_i^1}\,\overline{\chi_i^2} := \overline{\exists\bar{\alpha}_i^1.F_i^1}\ \overline{\exists\bar{\alpha}_i^2.F_i^2}$.

As seen above, we extend the notion of substitution to handle predicate variable atoms, where the replacement of each occurrence of a variable depends on the argument of that variable. When a variable occurs on the left-hand side of a substitution pair with a $+$ sign (resp. $-$ sign) in the upper index, when the substitution is applied, this pair contributes only to substituting a positive (resp. negative) occurrence of this variable. For example,

$$(x(t_1) \Rightarrow x(t_2) \wedge x(t_3))[x^+(t) := a(t);\ x^-(t) := b(t)] = (b(t_1) \Rightarrow a(t_2) \wedge a(t_3)).$$

3 The GADT Type System

τ will denote terms of sort s_{ty} in $\mathcal{L}$ below. Define the *type schemes* σ as $\forall\beta[\exists\bar{\alpha}.D].\beta$, where D is a conjunction of atoms in $\mathcal{L}$. Define the *pseudo type schemes* as either type schemes or $\forall\beta[\chi(\beta)].\beta$, where χ is a predicate variable and β is a variable of sort s_{ty}. A *simple environment* (or *monomorphic environment*) maps variables x to types τ. An *environment* (or *polymorphic environment*) maps variables x to type schemes σ, and a *pseudo environment* maps variables to pseudo type schemes. When a simple environment is appended to an environment, we identify τ and $\forall\beta[\beta \doteq \tau].\beta$ for $\beta \notin \text{FV}(\tau)$. When operations pertaining to formulas are applied to a (pseudo) type scheme $\forall\beta[\exists\bar{\alpha}.D].\beta$ or $\forall\beta[\chi(\beta)].\beta$, they are performed on the formula $\exists\bar{\alpha}.D$ or $\chi(\beta)$. When operations pertaining to (pseudo) type schemes (types) are applied to (pseudo, resp. simple) environments Γ, they are performed on the image of Γ. Define *environment fragments* Δ to be triples $\exists\bar{\alpha}[D].\Gamma$ of variables $\bar{\alpha}$, atomic conjunctions D in $\mathcal{L}$ and simple environment Γ.

Let $C \vDash D$ be a notational variant of $\vDash C \Rightarrow D$. Set $\Delta := \exists\bar{\beta}[D].\Gamma$ and $\Delta' := \exists\bar{\beta}'[D'].\Gamma'$ such that $\bar{\beta} \# \text{FV}(\Gamma')$, $\bar{\beta}' \# \text{FV}(\Delta)$ and $\bar{\beta}' \# C$. Let $C \vDash \Delta' \leqslant \Delta$ denote $C \wedge D' \vDash \exists\bar{\beta}.(D \wedge_{x \in \text{Dom}(\Gamma)} \Gamma(x) \doteq \Gamma'(x))$ when $\text{Dom}(\Gamma) = \text{Dom}(\Gamma')$, and otherwise a falsehood (compare lemma 3.5 of [12]). Let $\Delta \times \Delta'$ denote $\exists\bar{\beta}\bar{\beta}'[D \wedge D'].\Gamma \dot\cup \Gamma'$, and $\exists\bar{\beta}'[D']\Delta$ denote $\exists\bar{\beta}\bar{\beta}'[D \wedge D'].\Gamma$.

Proposition 3. *Properties of environment fragments (see [12] lemma 3.15).*

f-Hide. $\vDash \Delta \leqslant \exists\bar{\alpha}.\Delta$.

f-Imply. $C_1 \Rightarrow C_2 \vDash [C_1]\Delta \leqslant [C_2]\Delta$.

f-Enrich. $C \Rightarrow \Delta_1 \leqslant \Delta_2 \vDash [C]\Delta_1 \leqslant [C]\Delta_2$.

f-Ex. $\forall\bar{\alpha}.\Delta_1 \leqslant \Delta_2 \vDash (\exists\bar{\alpha}.\Delta_1) \leqslant (\exists\bar{\alpha}.\Delta_2)$.

f-And. $\Delta_1 \leqslant \Delta_2 \vDash \Delta \times \Delta_1 \leqslant \Delta \times \Delta_2$.

Unfortunately, our type inference algorithm does not handle disjunctive patterns. We therefore do not introduce them in our type system, but because they are very convenient, we remove them from pattern matching clauses by source code transformation in our implementation.

First, we present the type system in the standard, natural deduction style. The *type judgement* $C, \Gamma, \Sigma \vdash e\colon \tau$ or $C, \Gamma, \Sigma \vdash e\colon \sigma$ is composed of a formula C in $\mathcal{L}$, an environment Γ, a set of data constructors Σ, an expression e and a type τ or type scheme σ. A *pseudo type judgement* is composed of a formula C in $\mathcal{L}^{(\chi)}$, a pseudo environment Γ, a set of data constructors Σ, an expression e and a type τ or pseudo type scheme σ. The intended meaning of the type judgement $C, \Gamma, \Sigma \vdash e\colon \tau$ is: for every interpretation $\mathcal{I}$, ρ, if $\mathcal{I}, \rho \vDash C$, then the expression e has a ground type $\rho(\tau)$ in a ground environment $\rho(\mathcal{I}(\Gamma))$. We define validity of type judgements in table 2: $\bullet$, where D is a conjunction of atoms in $\mathcal{L}$.

Note that the lack of the standard type schemes $\forall\bar{\alpha}[D].\tau$ is only for the simplicity of presentation, as they are equivalent to $\forall\beta[\exists\bar{\alpha}.D \wedge \beta \doteq \tau].\beta$.

A *data constructor* K for a *datatype* ε (recall that the sort s_{ty} holds two categories of elements: datatypes and function types) has definition $K :: \forall\bar{\alpha}\bar{\beta}[D].\tau_1 \times \ldots \times \tau_n \to \varepsilon(\bar{\alpha})$ where $\text{FV}(D, \tau_1, \ldots, \tau_n) \subseteq \bar{\alpha}\bar{\beta}$. D is a solved form formula $\exists\bar{\beta}'.A$. Denote the set of data constructors by Σ.

- Patterns (syntax-directed)

p-Empty
$$C, \Sigma \vdash 0 : \tau \longrightarrow \exists \varnothing[\boldsymbol{F}]\{\}$$

p-Wild
$$C, \Sigma \vdash 1 : \tau \longrightarrow \exists \varnothing[\boldsymbol{T}]\{\}$$

p-And
$$\frac{\forall i \quad C, \Sigma \vdash p_i : \tau \longrightarrow \Delta_i}{C, \Sigma \vdash p_1 \wedge p_2 : \tau \longrightarrow \Delta_1 \times \Delta_2}$$

p-Var
$$C, \Sigma \vdash x : \tau \longrightarrow \exists \varnothing[\boldsymbol{T}]\{x \mapsto \tau\}$$

p-Cstr
$$\frac{\forall i \quad C \wedge D, \Sigma \vdash p_i : \tau_i \longrightarrow \Delta_i \quad \Sigma \ni K :: \forall \bar{\alpha} \bar{\beta}[D].\tau_1 \times ... \times \tau_n \to \varepsilon(\bar{\alpha}) \quad \bar{\beta} \# \mathrm{FV}(C)}{C, \Sigma \vdash K p_1 ... p_n : \varepsilon(\bar{\alpha}) \longrightarrow \exists \bar{\beta}[D](\Delta_1 \times ... \times \Delta_n)}$$

- Patterns (non-syntax-directed)

p-EqIn
$$\frac{C, \Sigma \vdash p : \tau' \longrightarrow \Delta \quad C \vDash \tau \doteq \tau'}{C, \Sigma \vdash p : \tau \longrightarrow \Delta}$$

p-SubOut
$$\frac{C, \Sigma \vdash p : \tau \longrightarrow \Delta' \quad C \vDash \Delta' \leqslant \Delta}{C, \Sigma \vdash p : \tau \longrightarrow \Delta}$$

p-Hide
$$\frac{C, \Sigma \vdash p : \tau \longrightarrow \Delta \quad \bar{\alpha} \# \mathrm{FV}(\tau, \Delta)}{\exists \bar{\alpha}.C, \Sigma \vdash p : \tau \longrightarrow \Delta}$$

- Expressions (syntax-directed)

Var
$$\frac{\Gamma(x) = \forall \beta[\exists \bar{\alpha}.D].\beta \quad C \vDash D}{C, \Gamma, \Sigma \vdash x : \beta}$$

Cstr
$$\frac{\forall i\, C, \Gamma, \Sigma \vdash e_i : \tau_i \qquad C \vDash D \quad \Sigma \ni K :: \forall \bar{\alpha} \bar{\beta}[D].\tau_1 ... \tau_n \to \varepsilon(\bar{\alpha})}{C, \Gamma, \Sigma \vdash K e_1 ... e_n : \varepsilon(\bar{\alpha})}$$

App
$$\frac{C, \Gamma, \Sigma \vdash e_1 : \tau' \to \tau \quad C, \Gamma, \Sigma \vdash e_2 : \tau'}{C, \Gamma, \Sigma \vdash e_1\, e_2 : \tau}$$

LetRec
$$\frac{C, \Gamma', \Sigma \vdash e_1 : \sigma \quad C, \Gamma', \Sigma \vdash e_2 : \tau \quad \sigma = \forall \beta[\exists \bar{\alpha}.D].\beta \quad \Gamma' = \Gamma\{x \mapsto \sigma\}}{C, \Gamma, \Sigma \vdash \mathbf{letrec}\, x = e_1\, \mathbf{in}\, e_2 : \tau}$$

Abs
$$\frac{\forall i\, C, \Gamma, \Sigma \vdash c_i : \tau_1 \to \tau_2}{C, \Gamma, \Sigma \vdash \lambda(c_1 ... c_n) : \tau_1 \to \tau_2}$$

- Expressions (non-syntax-directed)

Gen
$$\frac{C \wedge D, \Gamma, \Sigma \vdash e : \beta \quad \beta \bar{\alpha} \# \mathrm{FV}(\Gamma, C)}{C \wedge \exists \beta \bar{\alpha}.D, \Gamma, \Sigma \vdash e : \forall \beta[\exists \bar{\alpha}.D].\beta}$$

Inst
$$\frac{C, \Gamma, \Sigma \vdash e : \forall \bar{\alpha}[D].\tau' \quad C \vDash D[\bar{\alpha} := \bar{\tau}]}{C, \Gamma, \Sigma \vdash e : \tau'[\bar{\alpha} := \bar{\tau}]}$$

DisjElim
$$\frac{C, \Gamma, \Sigma \vdash e : \tau \qquad D, \Gamma, \Sigma \vdash e : \tau}{C \vee D, \Gamma, \Sigma \vdash e : \tau}$$

Hide
$$\frac{C, \Gamma, \Sigma \vdash e : \tau \quad \bar{\alpha} \# \mathrm{FV}(\Gamma, \tau)}{\exists \bar{\alpha}.C, \Gamma, \Sigma \vdash e : \tau}$$

Equ
$$\frac{C, \Gamma, \Sigma \vdash e : \tau \quad C \vDash \tau \doteq \tau'}{C, \Gamma, \Sigma \vdash e : \tau'}$$

FElim
$$\frac{}{\boldsymbol{F}, \Gamma, \Sigma \vdash e : \tau}$$

- Clauses

Clause
$$\frac{C, \Sigma \vdash p : \tau_1 \longrightarrow \exists \bar{\beta}[D]\Gamma' \quad C \wedge D, \Gamma\Gamma', \Sigma \vdash e : \tau_2 \quad \bar{\beta} \# \mathrm{FV}(C, \Gamma, \tau_2)}{C, \Gamma, \Sigma \vdash p.e : \tau_1 \to \tau_2}$$

Table 2. Typing rules

A closed expression e is *well typed* when $C, \epsilon, \Sigma \vdash e : \sigma$ holds for some satisfiable constraint C.

Proposition 4. *Constructor* $K :: \forall \bar{\alpha} \bar{\beta}[D].\tau_1 \times ... \times \tau_n \to \varepsilon(\bar{\alpha})$ *where* $D = \exists \bar{\beta}'.A$, *is equivalent to* $K :: \forall \bar{\alpha} \bar{\gamma_i}[\exists \bar{\beta} \bar{\beta}'.\bar{\gamma_i} \doteq \bar{\tau_i} \wedge A].\gamma_1 \times ... \times \gamma_n \to \varepsilon(\bar{\alpha})$.

Proposition 5. *Constructors of the form* $K :: \forall \bar{\alpha_i} \bar{\beta}[D].\tau_1 \times ... \times \tau_n \to \varepsilon(\bar{\alpha_i})$ *where* $D = \exists \bar{\beta}'.A$, *are equivalent to constructors of the form* $K :: \forall \alpha \bar{\beta}[\exists \bar{\alpha_i} \bar{\beta}'.\alpha \doteq \alpha_1 \to ... \to \alpha_m \wedge A].\gamma_1 \times ... \times \gamma_n \to \varepsilon(\alpha)$ *when all uses of* $\varepsilon(\tau_1, \ ..., \ \tau_m)$ *are translated to* $\varepsilon(\tau_1 \to ... \to \tau_m)$.

Lemma 6. Weakening *(patterns and expressions). Assume* $C_1 \vDash C_2$. *If* $C_2, \Sigma \vdash p : \tau \longrightarrow \Delta$ *(resp.* $C_2, \Gamma, \Sigma \vdash \mathrm{ce} : \tau$, $C_2, \Gamma, \Sigma \vdash \mathrm{ce} : \sigma$*) is derivable, then there exists a derivation of* $C_1, \Sigma \vdash p : \tau \longrightarrow \Delta$ *(resp.* $C_1, \Gamma, \Sigma \vdash \mathrm{ce} : \tau$, $C_1, \Gamma, \Sigma \vdash \mathrm{ce} : \sigma$*) of the same structure.*

The lemma follows from transitivity of $\vDash$ ($A \vDash B$ and $B \vDash C$ imply $A \vDash C$) by induction on the structure of the derivation.

Lemma 7. *If* $\Sigma \subset \Sigma'$ *and* $C, \Sigma \vdash p : \tau \longrightarrow \Delta$ *(resp.* $C, \Gamma, \Sigma \vdash \mathrm{ce} : \tau$, $C, \Gamma, \Sigma \vdash \mathrm{ce} : \sigma$*) is derivable, then there exists a derivation of* $C, \Sigma' \vdash p : \tau \longrightarrow \Delta$ *(resp.* $C, \Gamma, \Sigma' \vdash \mathrm{ce} : \tau$, $C, \Gamma, \Sigma' \vdash \mathrm{ce} : \sigma$*) of the same structure.*

Now, we present pseudo type judgements declaratively by reducing them to constraints. For $\bar{c} = \overline{p_i.e_i}$, $[\![\Gamma, \Sigma \vdash \bar{c} : \tau_1 \to \tau_2]\!] := \wedge_i [\![\Gamma, \Sigma \vdash p_i.e_i : \tau_1 \to \tau_2]\!]$. (The presentation is a little bit heavy due to explicit capture-avoidance conditions.)

- Patterns (constraint generation)

$$
\begin{array}{rcl}
[\![\Sigma \vdash 0{\downarrow}\tau]\!] & = & \boldsymbol{T} \\
[\![\Sigma \vdash 1{\downarrow}\tau]\!] & = & \boldsymbol{T} \\
[\![\Sigma \vdash x{\downarrow}\tau]\!] & = & \boldsymbol{T} \\
[\![\Sigma \vdash p_1 \wedge p_2{\downarrow}\tau]\!] & = & [\![\Sigma \vdash p_1{\downarrow}\tau]\!] \wedge [\![\Sigma \vdash p_2{\downarrow}\tau]\!] \\
[\![\Sigma \vdash K p_1 ... p_n{\downarrow}\tau]\!] & = & \exists \bar{\alpha}'.(\varepsilon(\bar{\alpha}') \doteq \tau \wedge \forall \bar{\beta}'.D[\bar{\alpha}\bar{\beta} := \bar{\alpha}'\bar{\beta}'] \Rightarrow \wedge_i [\![p_i{\downarrow}\tau_i[\bar{\alpha}\bar{\beta} := \bar{\alpha}'\bar{\beta}']]\!]) \\
 & & \text{where } \Sigma \ni K :: \forall \bar{\alpha}\bar{\beta}[D].\tau_1 \times ... \times \tau_n \to \varepsilon(\bar{\alpha}), \bar{\alpha}'\bar{\beta}' \# \mathrm{FV}(\Sigma, \tau)
\end{array}
$$

- Patterns (environment fragment generation)

$$
\begin{array}{rcl}
[\![\Sigma \vdash 0{\uparrow}\tau]\!] & = & \exists \varnothing[\boldsymbol{F}]\{\} \\
[\![\Sigma \vdash 1{\uparrow}\tau]\!] & = & \exists \varnothing[\boldsymbol{T}]\{\} \\
[\![\Sigma \vdash x{\uparrow}\tau]\!] & = & \exists \varnothing[\boldsymbol{T}]\{x \mapsto \tau\} \\
[\![\Sigma \vdash p_1 \wedge p_2{\uparrow}\tau]\!] & = & [\![\Sigma \vdash p_1{\uparrow}\tau]\!] \times [\![\Sigma \vdash p_2{\uparrow}\tau]\!] \\
[\![\Sigma \vdash K p_1 ... p_n{\uparrow}\tau]\!] & = & \exists \bar{\alpha}'\bar{\beta}'[\varepsilon(\bar{\alpha}') \doteq \tau \wedge D[\bar{\alpha}\bar{\beta} := \bar{\alpha}'\bar{\beta}']](\times_i [\![p_i{\uparrow}\tau_i[\bar{\alpha}\bar{\beta} := \bar{\alpha}'\bar{\beta}']]\!]) \\
 & & \text{where } \Sigma \ni K :: \forall \bar{\alpha}\bar{\beta}[D].\tau_1 \times ... \times \tau_n \to \varepsilon(\bar{\alpha}), \bar{\alpha}'\bar{\beta}' \# \mathrm{FV}(\Sigma, \tau) \\
[\![\Sigma \vdash K p_1 ... p_n{\uparrow}\tau]\!] & = & \exists \varnothing[\boldsymbol{F}]\{\} \quad \text{when} \quad K \notin \mathrm{Dom}(\Sigma)
\end{array}
$$

Table 3. Type inference for patterns

$$
\begin{array}{rcl}
[\![\Gamma, \Sigma \vdash x\colon \tau]\!] & = & \boldsymbol{F} \quad \text{when} \quad x \notin \mathrm{Dom}(\Gamma) \\
[\![\Gamma, \Sigma \vdash x\colon \tau]\!] & = & \exists \beta'\bar{\alpha}'.(D[\beta\bar{\alpha} := \beta'\bar{\alpha}'] \wedge \beta' \doteq \tau) \\
 & & \text{where } \Gamma(x) = \forall \beta[\exists \bar{\alpha}.D].\beta, \beta'\bar{\alpha}' \# \mathrm{FV}(\Gamma, \Sigma, \tau) \\
[\![\Gamma, \Sigma \vdash \lambda \bar{c}\colon \tau]\!] & = & \exists \alpha_1\alpha_2.([\![\Gamma, \Sigma \vdash \bar{c}\colon \alpha_1 \to \alpha_2]\!] \wedge \alpha_1 \to \alpha_2 \doteq \tau), \alpha_1\alpha_2 \# \mathrm{FV}(\Gamma, \Sigma, \tau) \\
[\![\Gamma, \Sigma \vdash e_1\, e_2\colon \tau]\!] & = & \exists \alpha.([\![\Gamma, \Sigma \vdash e_1\colon \alpha \to \tau]\!] \wedge [\![\Gamma, \Sigma \vdash e_2\colon \alpha]\!]), \alpha \# \mathrm{FV}(\Gamma, \Sigma, \tau) \\
[\![\Gamma, \Sigma \vdash K e_1 ... e_n\colon \tau]\!] & = & \boldsymbol{F} \quad \text{when} \quad K \notin \mathrm{Dom}(\Sigma) \\
[\![\Gamma, \Sigma \vdash K e_1 ... e_n\colon \tau]\!] & = & \exists \bar{\alpha}'\bar{\beta}'.(\wedge_i [\![\Gamma, \Sigma \vdash e_i\colon \tau_i[\bar{\alpha}\bar{\beta} := \bar{\alpha}'\bar{\beta}']]\!] \wedge D[\bar{\alpha}\bar{\beta} := \bar{\alpha}'\bar{\beta}'] \wedge \varepsilon(\bar{\alpha}') \doteq \tau) \\
 & & \text{where } \Sigma \ni K :: \forall \bar{\alpha}\bar{\beta}[D].\tau_1 \times ... \times \tau_n \to \varepsilon(\bar{\alpha}), \bar{\alpha}'\bar{\beta}' \# \mathrm{FV}(\Gamma, \Sigma, \tau) \\
[\![\Gamma, \Sigma \vdash \textbf{letrec}\, x = e_1\, \textbf{in}\, e_2\colon \tau]\!] & = & (\forall \beta(\chi(\beta) \Rightarrow [\![\Gamma\{x \mapsto \forall \beta[\chi(\beta)].\beta\}, \Sigma \vdash e_1\colon \beta]\!])) \wedge \\
 & & (\exists \alpha.\chi(\alpha)) \wedge [\![\Gamma\{x \mapsto \forall \beta[\chi(\beta)].\beta\}, \Sigma \vdash e_2\colon \tau]\!] \\
 & & \text{where } \beta \# \mathrm{FV}(\Gamma, \Sigma, \tau), \chi \# \mathrm{PV}(\Gamma, \Sigma) \\
[\![\Gamma, \Sigma \vdash p.e\colon \tau_1 \to \tau_2]\!] & = & [\![\Sigma \vdash p{\downarrow}\tau_1]\!] \wedge \forall \bar{\beta}.D \Rightarrow [\![\Gamma\Gamma', \Sigma \vdash e\colon \tau_2]\!] \\
 & & \text{where } \exists \bar{\beta}[D]\Gamma' \text{ is } [\![\Sigma \vdash p{\uparrow}\tau_1]\!], \bar{\beta} \# \mathrm{FV}(\Gamma, \Sigma, \tau_2) \\
[\![\Gamma, \Sigma \vdash \mathrm{ce}\colon \forall \bar{\alpha}[D].\tau]\!] & = & \forall \bar{\alpha}'.D[\bar{\alpha} := \bar{\alpha}'] \Rightarrow [\![\Gamma, \Sigma \vdash \mathrm{ce}\colon \tau[\bar{\alpha} := \bar{\alpha}']]\!], \bar{\alpha}' \# \mathrm{FV}(\Gamma, \Sigma)
\end{array}
$$

Table 4. Type inference for expressions and clauses

The two presentations are equivalent, in the sense of theorems *correctness* and *completeness* below.

Lemma 8. Correctness *(patterns).* $[\![\Sigma \vdash p{\downarrow}\tau]\!] \vdash p\colon \tau \longrightarrow [\![\Sigma \vdash p{\uparrow}\tau]\!]$.

Theorem 9. Correctness *(expressions).* $[\![\Gamma, \Sigma \vdash \mathrm{ce}\colon \tau]\!], \Gamma, \Sigma \vdash \mathrm{ce}\colon \tau$.

$\Gamma' \doteq \Gamma''$ stands for $\forall x \in \mathrm{Dom}(\Gamma') \cup \mathrm{Dom}(\Gamma'').\Gamma'(x) \doteq \Gamma''(x)$ and is false when $\mathrm{Dom}(\Gamma') \neq \mathrm{Dom}(\Gamma'')$. Recall that for $\Delta := \exists \bar{\beta}[D].\Gamma$ and $\Delta' := \exists \bar{\beta}'[D'].\Gamma'$ such that $\bar{\beta} \# \mathrm{FV}(\Gamma')$, $\bar{\beta}' \# \mathrm{FV}(\Delta)$ and $\bar{\beta}' \# C$, $C \vDash \Delta' \leqslant \Delta$ denotes $C \wedge D' \vDash \exists \bar{\beta}.D \wedge \Gamma \doteq \Gamma'$. Observe, that $C \vDash \Delta' \leqslant \Delta$ iff $C \vDash \forall \bar{\beta}'.D' \Rightarrow \exists \bar{\beta}.D \wedge \Gamma \doteq \Gamma'$.

Lemma 10. Completeness *(patterns). Let* $\Delta = \exists \bar{\beta}'[D']\Gamma'$ *and* $[\![\Sigma \vdash p{\uparrow}\tau]\!] = \exists \bar{\beta}''[D'']\Gamma'' = \Delta'$. $C, \Sigma \vdash p\colon \tau \longrightarrow \Delta$ *implies* $C \vDash [\![\Sigma \vdash p{\downarrow}\tau]\!]$ *and* $C \vDash \forall \bar{\beta}''.D'' \Rightarrow \exists \bar{\beta}'.(D' \wedge \Gamma'' \doteq \Gamma')$, *i.e.* $C \vDash \Delta' \leqslant \Delta$.

Lemma 11. *Let* Γ *be an environment and* Γ', Γ'' *be simple (i.e. monomorphic) environments. For any* e, τ, $C \wedge \Gamma' \doteq \Gamma'', \Gamma\Gamma', \Sigma \vdash e\colon \tau$ *iff* $C \wedge \Gamma' \doteq \Gamma'', \Gamma\Gamma'', \Sigma \vdash e\colon \tau$.

Theorem 12. Completeness *(expressions). If* $\mathrm{PV}(C, \Gamma, \Sigma) = \varnothing$ *and* $C, \Gamma, \Sigma \vdash \mathrm{ce}\colon \tau$, *then there exists an interpretation of predicate variables* $\mathcal{I}$ *such that* $\mathcal{I}, C \vDash [\![\Gamma, \Sigma \vdash \mathrm{ce}\colon \tau]\!]$.

Corollary 13. *If* $C, \Gamma, \Sigma \vdash \text{ce}: \forall\bar{\alpha}[D].\tau$ *and* $\bar{\alpha}\#\text{FV}(\Gamma, \Sigma)$*, then there is an interpretation* $\mathcal{I}$ *such that* $\mathcal{I}$*,* $C \vDash \forall\bar{\alpha}.D \Rightarrow [\![\Gamma, \Sigma \vdash \text{ce}: \tau]\!]$.

3.1 Example

Consider a function `eval` defined by cases over a datatype Term(α) with constructors `Lit`$:: \forall\alpha[\alpha \doteq \text{Int}].\text{Int} \to \text{Term}(\alpha)$, `IsZero`: $\forall\alpha[\alpha \doteq \text{Bool}].\text{Term}(\text{Int}) \to \text{Term}(\alpha)$, `If`: $\forall\alpha.\text{Term}(\text{Bool}) \to \text{Term}(\alpha) \to \text{Term}(\alpha) \to \text{Term}(\alpha)$, and functions $\Gamma = \{\text{eq} \mapsto \forall\alpha.\alpha \to \alpha \to \text{Bool}; \text{ite} \mapsto \forall\alpha.\text{Bool} \to \alpha \to \alpha \to \alpha; \text{zero} \mapsto \text{Int}\}$, defined by an expression e:

$$\begin{array}{ll} \textbf{letrec} & \text{eval} = \\ & \quad \lambda(\text{Lit}\, x.x, \text{IsZero}\, y.\text{eq}\,(\text{eval}\, y)\,\text{zero}, \\ & \qquad \text{If}\, y_1\, y_2\, y_3.\text{ite}\,(\text{eval}\, y_1)\,(\text{eval}\, y_2)\,(\text{eval}\, y_3)) \\ \textbf{in} & \text{eval} \end{array}$$

Let $\Gamma' = \Gamma\{x \mapsto \forall\beta[\chi(\beta)].\beta\}$. The constraint derived is:

$$\begin{array}{rcl} [\![\Gamma \vdash e: \alpha_{\text{res}}]\!] & = & (\forall\beta(\chi(\beta) \Rightarrow \exists\alpha_1\alpha_2.([\![\Gamma' \vdash \bar{c}: \alpha_1 \to \alpha_2]\!] \wedge \\ & & \alpha_1 \to \alpha_2 \doteq \beta))) \wedge \\ & & (\exists\alpha.\chi(\alpha)) \wedge \exists\beta''.\chi(\beta'') \wedge \beta'' \doteq \alpha_{\text{res}} \end{array}$$

After normalization to alternation-minimizing prenex normal form:

$$\begin{array}{l} \exists\alpha_4\beta''\forall\beta\exists\alpha_1\alpha_2\exists\alpha_1'\alpha_2'\alpha_3'\forall\beta_1\beta_2\exists\alpha\alpha'\alpha''\alpha'''\beta'.... \\ \chi(\alpha) \wedge \chi(\beta'') \wedge \beta'' \doteq \alpha_{\text{res}} \wedge \\ (\chi(\beta) \Rightarrow \text{Term}(\alpha_1') \doteq \alpha_1 \wedge \text{Term}(\alpha_2') \doteq \alpha_1 \wedge \\ \text{Term}(\alpha_3') \doteq \alpha_1 \wedge \alpha_1 \to \alpha_2 \doteq \beta) \wedge \\ (\chi(\beta) \wedge \text{Term}(\beta_1) \doteq \alpha_1 \wedge \beta_1 \doteq \text{Int} \Rightarrow \text{Int} \doteq \alpha_2) \wedge \\ (\chi(\beta) \wedge \text{Term}(\beta_2) \doteq \alpha_1 \wedge \beta_2 \doteq \text{Bool} \Rightarrow \\ \quad \alpha'' \to \alpha'' \to \text{Bool} \doteq \alpha' \to \alpha \to \alpha_2 \wedge \chi(\beta') \wedge \beta' \doteq \alpha''' \to \\ \alpha' \wedge \text{Term}(\text{Int}) \doteq \alpha''' \wedge \text{Int} \doteq \alpha) \wedge \\ (\chi(\beta) \wedge ... \Rightarrow ...) \end{array}$$

4 Existential Types

We add existentially quantified types to our type system, in a nominal (i.e. "identified by name") way. Instead of extending the language of types, we extend the language of expressions to provide place for automatic definition of GADTs that encapsulate the existentially quantified types. When $K :: \forall\bar{\alpha}\bar{\beta}\gamma[E].\gamma \to \varepsilon_K(\bar{\alpha}) \in \Sigma$ is the only data constructor for type $\varepsilon_K(\bar{\alpha})$, the pretty-printer for types prints $\varepsilon_K(\bar{\tau})$ as $(\exists\bar{\beta}\gamma[E[\bar{\alpha} := \bar{\tau}]].\gamma)$ (or $(\exists\bar{\beta}[E[\bar{\alpha} := \bar{\tau}]].\tau_e)$ when $\gamma \doteq \tau_e \in E$). We add several syntax-directed rules: "type-level existential quantifier" introduction `ExCases` and elimination `ExLetIn`, the scoping rule `CstrIntro`, and their corresponding abstract syntax expressions: $\lambda[K]\bar{c}$, **let** $p = e_1$ **in** e_2 and **extype** $K.e$ for a constructor K, pattern matching clauses $\bar{c}$ and expressions e, e_1, e_2. The constructor K in $\lambda[K].\bar{c}$ and **extype** $K.e$ will not appear in concrete syntax, instead a unique identifier will be provided by the parser for a coupling of **extype** $K.e$ and $\lambda[K].\bar{c}$ resulting from various forms of "syntactic sugar".

By Dom(Σ) we denote the set of identifiers of the constructor definitions in Σ.

`CstrIntro`

$$\frac{C, \Gamma, \Sigma \cup \{K :: \forall\alpha\gamma[E].\gamma \to \varepsilon_K(\alpha)\} \vdash e: \tau \quad K \notin \text{Dom}(\Sigma)}{C, \Gamma, \Sigma \vdash \textbf{extype}\ K.e: \tau}$$

`ExCases`

$$\frac{\begin{array}{ll} & K :: \forall\alpha\gamma[E].\gamma \to \varepsilon_K(\alpha) \in \Sigma \quad \bar{\beta}_i\#\text{FV}(C, \Gamma, \tau_i') \qquad \text{FV}(E) \subseteq \alpha\gamma \\ (\forall i) & C, \Sigma \vdash p_i: \tau \longrightarrow \exists\bar{\beta}_i[D_i]\Gamma_i' \quad C \wedge D_i, \Gamma\Gamma_i', \Sigma \vdash e_i: \tau_i' \quad C \wedge D_i \vDash E[\gamma := \tau_i'] \end{array}}{C, \Gamma, \Sigma \vdash \lambda[K](p_1.e_1 ... p_n.e_n): \tau \to \varepsilon_K(\alpha)}$$

`ExLetIn`

$$\frac{\begin{array}{l} K :: \forall\alpha\gamma[E].\gamma \to \varepsilon_K(\alpha) \in \Sigma \quad \bar{\beta}'\#\text{FV}(C, \Gamma, \tau) \\ C \wedge E, \Sigma \vdash p: \gamma \longrightarrow \exists\bar{\beta}'[D]\Gamma' \quad C, \Gamma, \Sigma \vdash e_1: \varepsilon_K(\alpha) \quad C \wedge D \wedge E, \Gamma\Gamma', \Sigma \vdash e_2: \tau \end{array}}{C, \Gamma, \Sigma \vdash \textbf{let}\ p = e_1\ \textbf{in}\ e_2: \tau}$$

Table 5. Added typing rules

Note that by proposition 5, we do not lose generality (expressivity) by using single-argument datatypes $\varepsilon_K(\alpha)$ rather than the general form $\varepsilon_K(\bar{\alpha})$. We will denote the encoding $\alpha_1 \to ... \to \alpha_m$ of $\bar{\alpha_i}$ as $\vec{\alpha_i}$.

Proposition 14. Interpretation of extension. *Let* C, Γ, Σ, ce, τ *be such that* C, Γ, $\Sigma \vdash \text{ce}: \tau$ *holds in the type system extended with* `CstrIntro`*,* `ExCases` *and* `ExLetIn`*, and let* ce′ *be* ce *with subexpressions* **extype** $K.e$ *replaced by* e*, subexpressions* $\lambda[K](p_1.e_1 ... p_n.e_n)$ *replaced by* $\lambda(p_1.K\, e_1' ... p_n.K\, e_n')$*, and subexpressions* **let** $p = e_1$ **in** e_2 *replaced by* $\lambda(K p.e_2')\ e_1'$*, where* C', Γ', $\Sigma' \vdash e_1: \varepsilon_K(\tau)$ *is the corresponding subtree for* e_1 *in the derivation of* C, Γ, $\Sigma \vdash \text{ce}: \tau$*;* e_i' *are* e_i *with the replacement applied recursively. Then* C, Γ, $\Sigma_o \vdash \text{ce}': \tau$ *holds in the original type system where* Σ_o *is* Σ *with additional data constructors* K*, introduced by* `CstrIntro` *in* $C, \Gamma, \Sigma \vdash \text{ce}: \tau$.

The extended type system is a conservative extension of the base type system since all added rules are driven by new syntactic constructs. But we would like a form of conservativity expressed by a reverse claim to proposition 14: can we use existential abstraction (`ExCases` instead of `Cases`) without fear of losing expressivity?

Take an expression ce in the type system extended with `CstrIntro`, `ExCases` and `ExLetIn`, such that for every $\lambda[K]$ in ce there is an **extype** K in ce and there are no more occurrences of K in ce (there is at most one occurrence of $\lambda[K]$ for any K, and **extype**-bound K is not used otherwise), and to keep things simple, all occurrences of **extype** K in ce form a prefix (i.e. are at the root). Form expresions ce′ and ce″ as follows.

Let let ce′ be as in proposition 14: ce with subexpressions **extype** $K.e$ replaced by e, subexpressions $\lambda[K](p_1.e_1 ... p_n.e_n)$ replaced by $\lambda(p_1.K\, e_1' ... p_n.K\, e_n')$, and subexpressions **let** $p = e_1$ **in** e_2 replaced by $\lambda(?_K\, p.e_2')\ e_1'$,

where $?_K$ will be determined later; e_i' are e_i with the replacement applied recursively. Let Σ_u, given Σ and $\overline{\tau_K}$, be Σ with additional constructors $K :: \forall\alpha\gamma[\exists\overline{\mathrm{FV}(\tau_K)}.\alpha \doteq \overrightarrow{\mathrm{FV}(\tau_K)} \wedge \gamma \doteq \tau_K].\gamma \to \varepsilon_K(\alpha)$, for all K occurring in **extype** K in ce. Let $?_K$ be one of **extype**-bound constructors such that $C, \Gamma, \Sigma_u \vdash \mathrm{ce}' : \tau'$ holds for some type τ', or arbitrary constructors otherwise. Note that $?_K$ is well defined in that if the choice of constructors is not arbitrary, then it does not depend on the type τ' and derivation of $C, \Gamma, \Sigma_u \vdash \mathrm{ce}' : \tau'$.

Let ce″ be ce with subexpressions **extype** $K.e$ replaced by e, subexpressions $\lambda[K](p_1.e_1 \ldots p_n.e_n)$ replaced by $\lambda(p_1.e_1'' \ldots p_n.e_n'')$, and subexpressions **let** $p = e_1$ **in** e_2 replaced by $\lambda(p.e_2'')\ e_1''$, where e_i'' are e_i with the replacement applied recursively. Let $u(\tau)$ be a function that replaces subterms $\varepsilon_K(.)$ by τ_K, for τ_K the same as in Σ_u.

Proposition 15. Conservative Translation. *Let* $\mathrm{ce}, C, \Gamma, \Sigma, \tau, \overline{\tau_K}$ *be such that* $C, \Gamma, \Sigma_u \vdash \mathrm{ce}' : \tau$ *holds in the original type system. Then there is* u *such that* $C, \Gamma, \Sigma \vdash \mathrm{ce}'' : u(\tau)$ *holds in the original type system, and* $C, \Gamma, \Sigma \vdash \mathrm{ce} : \tau$ *holds in the type system extended with* `CstrIntro`, `ExCases` *and* `ExLetIn`, *with applications of* `CstrIntro` *introducing constructors* $K :: \forall\alpha\gamma[\exists\overline{\mathrm{FV}(\tau_K)}.\alpha \doteq \overrightarrow{\mathrm{FV}(\tau_K)} \wedge \gamma \doteq \tau_K].\gamma \to \varepsilon_K(\alpha)$ *for* $\bar{K} \# \mathrm{Dom}(\Sigma)$, *as in* Σ_u.

4.1 Type Inference Constraints for Existential Types

The type inference uses predicate variables to determine the existential guard in `ExCases`. We introduce a syntactic marker $()^i_K$ to point to constraint for branch i of introducing the existential type ε_K.

$$
\begin{array}{rcl}
[\![\Gamma, \Sigma \vdash \mathbf{extype}\, K.e : \tau]\!] & = & [\![\Gamma, \Sigma \cup \{K :: \forall\alpha\gamma[\chi_K(\alpha,\gamma)].\gamma \to \varepsilon_K(\alpha)\} \vdash e : \tau]\!] \\
 & & \text{where } \chi_K \notin \mathrm{PV}(\Gamma, \Sigma), K \notin \mathrm{Dom}(\Sigma) \\
[\![\Gamma, \Sigma \vdash \lambda[K]\overline{p_i.e_i} : \tau]\!] & = & \boldsymbol{F} \quad \text{when} \quad K \notin \mathrm{Dom}(\Sigma) \\
[\![\Gamma, \Sigma \vdash \lambda[K]\overline{p_i.e_i} : \tau]\!] & = & \exists\alpha_1\alpha_2.(\alpha_1 \to \varepsilon_K(\alpha_2) \doteq \tau) \wedge \\
 & & \wedge_i([\![\Sigma \vdash p_i{\downarrow}\alpha_1]\!] \wedge \\
 & & \quad \forall\bar{\beta}_i'.D_i \Rightarrow (\exists\alpha_3^i.[\![\Gamma\Gamma_i', \Sigma \vdash e_i : \alpha_3^i]\!] \wedge \chi_K(\alpha_2, \alpha_3^i))^i_K) \\
 & & \text{where } \exists\bar{\beta}_i'[D_i]\Gamma_i' \text{ is } [\![\Sigma \vdash p_i{\uparrow}\alpha_1]\!], \\
 & & \alpha_1\overline{\alpha_3^i}\,\bar{\beta}_i' \# \mathrm{FV}(\Gamma, \Sigma, \tau, \overline{D_i}), K :: \forall\alpha\gamma[\chi_K(\alpha,\gamma)].\gamma \to \varepsilon_K(\alpha) \in \Sigma \\
[\![\Gamma, \Sigma \vdash \mathbf{let}\, p = e_1\, \mathbf{in}\, e_2 : \tau]\!] & = & \exists\alpha_0\alpha_2.(\vee_{(\varepsilon_K,\chi_K)\in\mathcal{E}}\varepsilon_K(\alpha_2) \doteq \alpha_0) \wedge [\![\Gamma, \Sigma \vdash e_1 : \alpha_0]\!] \wedge \\
 & & \forall\alpha_3.((\vee_{(\varepsilon_K,\chi_K)\in\mathcal{E}}(\varepsilon_K(\alpha_2) \doteq \alpha_0 \wedge \chi_K(\alpha_2,\alpha_3))) \Rightarrow \\
 & & \quad ([\![\Sigma \vdash p{\downarrow}\alpha_3]\!] \wedge (\forall\bar{\beta}'.D \Rightarrow [\![\Gamma\Gamma', \Sigma \vdash e_2 : \tau]\!]))) \\
 & & \text{where } \exists\bar{\beta}'[D]\Gamma' \text{ is } [\![\Sigma \vdash p{\uparrow}\alpha_3]\!], \alpha_0\alpha_2\alpha_3\bar{\beta}' \# \mathrm{FV}(\Gamma, \Sigma, \tau), \\
 & & \mathcal{E} = \{\varepsilon_K, \chi_K | K :: \forall\alpha\gamma[\chi_K(\alpha,\gamma)].\gamma \to \varepsilon_K(\alpha) \in \Sigma\}
\end{array}
$$

Table 6. Type inference for the added expressions

Theorem 16. *Theorems 9 (Correctness) and 12 (Completeness) hold for the type system extended with* `CstrIntro`, `ExCases` *and* `ExLetIn`.

5 Deriving the Invariants

We assume that we have a *complete abduction algorithm* $\mathrm{Abd}(\mathcal{Q}, \overline{D_i, C_i})$ for JCAQP$_\mathcal{M}$ at our disposal (where D_i and C_i are conjunctions of atoms in $\mathcal{L}$), that generates a sequence of quantified conjunctions of atoms $\overline{\exists\bar{\alpha}_j.A_j} \subset \mathcal{F}$ (that is, each A_j meets the solved form property). Each answer $\exists\bar{\alpha}.A \in \mathrm{Abd}(\mathcal{Q}, \overline{D_i, C_i})$ meets the relevance condition: $\mathcal{M} \models \wedge_i(D_i \wedge A \Rightarrow C_i)$, validity condition: $\mathcal{M} \models \forall\bar{\alpha}\mathcal{Q}.A$, and consistency condition: $\mathcal{M} \models \wedge_i\forall\bar{\alpha}\exists(\bar{\alpha}^c).D_i \wedge A$. Completeness is understood as: for every JCAQP$_\mathcal{M}$ answer $\exists\bar{\alpha}_s.A_s$, there is $\exists\bar{\alpha}.A \in \mathrm{Abd}(\mathcal{Q}, \overline{D_i, C_i})$ and some $\bar{t}$ such that $\mathcal{M} \models A_s \Rightarrow A[\bar{\alpha} := \bar{t}]$ (with variables renamed so that $\bar{\alpha}_s \# \mathrm{FV}(A)$). In practice the algorithm will fall short of completeness; for more information consult [16].

Faced with an inference problem Φ, we solve it using iterated abduction. Let

$$\exists\delta\Phi \equiv \exists\delta\mathcal{Q}.\wedge_i(D_i \Rightarrow C_i) = \exists\delta\mathrm{NF}(\Phi)$$

where D_i, C_i are conjunctions of atoms, be quantifier alternation-minimizing normalization of Φ (the variable δ is used just to bring existentially quantified variables to the front, if possible).

We will say that a quantifier prefix and a conjunction of atoms $\mathcal{Q}.A$ are in *atomized form*, when there are no properties expressed in A using universal quantification that can also be expressed without it: if there are variables $\exists\bar{\alpha}$ to the left of $\forall\bar{\beta}$ in $\mathcal{Q}$ and atoms $B \subset A$ such that $\models(\exists\bar{\alpha}\forall\bar{\beta}.B) \equiv (\exists\bar{\alpha}.C) \wedge (\forall\bar{\beta}.D)$ for some conjunctions of atoms C, D with $\mathrm{FV}(C)\#\bar{\beta}$, then $B = B_1 \cup B_2$ with $\models(\exists\bar{\alpha}\forall\bar{\beta}.B) \equiv (\exists\bar{\alpha}.B_1) \wedge (\forall\bar{\beta}.B_2)$ and $\mathrm{FV}(B_1)\#\bar{\beta}$. We assume there is an algorithm $\mathrm{AF}(\mathcal{Q}.A)$ that for a conjunction of atoms returns an equivalent conjunction of atoms but in atomized form.

For a conjunction of atoms $A \in \mathcal{L}$, a quantifier prefix $\mathcal{Q}$, and variables $\bar{\alpha}$, $\overline{\bar{\beta}^\chi}$, where χ varies over $\mathrm{X}(\mathcal{Q})$, let $\mathrm{Split}\big(\mathcal{Q},\ \bar{\alpha},\ A,\ \overline{\bar{\beta}^\chi}\big) := \mathrm{Split}\big(\mathcal{Q},\ \bar{\alpha},\ \mathrm{AF}\big(\mathcal{Q}\big[\forall\overline{\bar{\beta}^\chi} := \exists\overline{\bar{\beta}^\chi}\,\big].A\big),\ \overline{\bar{\beta}^\chi},\ \overline{\top}\big)$ and let $\mathrm{Split}\big(\mathcal{Q}, \bar{\alpha}, A, \overline{\bar{\beta}^\chi}, \overline{A^0_\chi}\big)$ be a set generated by the following algorithm: table 7 – where $\mathrm{Strat}(A, \bar{\beta}^\chi)$ is computed as follows: for every $c \in A$, and for every $\beta_2 \in \mathrm{FV}(c)$ such that $\beta_1 <_\mathcal{Q} \beta_2$ for $\beta_1 \in \bar{\beta}^\chi$, if β_2 is universally quantified in $\mathcal{Q}$, then return $\bot$; otherwise, introduce a fresh variable α_f, replace $c := c[\beta_2 := \alpha_f]$, add $\beta_2 \doteq \alpha_f$ to A^R_χ and α_f to $\bar{\alpha}^\chi_{\mathrm{LR}}$, after replacing all such β_2 add the resulting c to A^L_χ.

$$
\begin{array}{rcl}
\alpha \prec \beta & \equiv & \alpha <_{\mathcal{Q}} \beta \vee \big(\alpha \leqslant_{\mathcal{Q}} \beta \wedge \alpha \in \overline{\bar{\beta}^{\chi}} \wedge \beta \notin \overline{\bar{\beta}^{\chi}}\big) \\
A^1_\chi & = & \Big\{c \in A \Big| \bar{\beta}^{\chi} \cap \max_{\prec}(\mathrm{FV}(c)) \neq \varnothing\Big\} \\
A^{+\max}_\chi & = & \Big\{c \in A \Big| \exists \beta_1\beta_2 \subset \mathrm{FV}(c).c \neq \beta_2 \doteq \beta_1 \wedge \beta_1 \prec \beta_2 \wedge \\
 & & \qquad \beta_1 \in \max_{\prec}(\bar{\beta}^{\chi} \cap \mathrm{FV}(c))\Big\} \\
A^{+\min}_\chi & = & \{c \in A | c \in A^{+\max}_\chi \wedge \not\models \mathcal{Q}.c\} \\
\text{for all } \overline{A^+_\chi} \text{ such that} & & \wedge_\chi (A^{+\min}_\chi \subset A^+_\chi \subset A^{+\max}_\chi \wedge A^+_\chi \text{ minimal w.r.t. } \subset) \wedge \\
 & & \models \mathcal{Q}.A \setminus \cup_\chi (A^1_\chi \cup A^+_\chi): \\
\text{if} & & \mathrm{Strat}(A^+_\chi, \bar{\beta}^{\chi}) \text{ returns } \bot \text{ for some } \chi \\
\text{then return} & & \bot \\
\text{else } \bar{\alpha}^{\chi}_+, A^L_\chi, A^R_\chi & = & \mathrm{Strat}(A^+_\chi, \bar{\beta}^{\chi}) \\
A_\chi & = & A^0_\chi \cup A^1_\chi \cup A^L_\chi \\
A_+ & = & \cup_\chi A^R_\chi \\
A_{\mathrm{res}} & = & A_+ \cup \widetilde{A_+}(A \setminus \cup_\chi (A^1_\chi \cup A^+_\chi)) \\
\bar{\alpha}^{\chi}_0 & = & \bar{\alpha} \cap \mathrm{FV}(A_\chi) \setminus \mathrm{FV}(\{c \in A | \exists \beta \in \mathrm{FV}(c).\beta <_{\mathcal{Q}} \beta^{\chi}\}) \text{ where } \beta^{\chi} \in \bar{\beta}^{\chi} \\
\bar{\alpha}^{\chi} & = & \left(\bar{\alpha}^{\chi}_0 \setminus \bigcup_{\chi' <_{\mathcal{Q}} \chi} \bar{\alpha}^{\chi'}_0\right)\bar{\alpha}^{\chi}_+ \\
\text{if} & & \cup_\chi (\bar{\alpha}^{\chi} \setminus \bar{\beta}^{\chi}) \neq \varnothing \\
\text{then } \mathcal{Q}', \overline{\bar{\alpha}^{\chi}_+}, A'_{\mathrm{res}}, \overline{\exists \bar{\alpha}'^{\chi}.A'_\chi} & = & \mathrm{Split}\big(\mathcal{Q}\big[\overline{\forall \bar{\beta}^{\chi}} := \overline{\forall(\bar{\beta}^{\chi} \cup \bar{\alpha}^{\chi})}\big], \bar{\alpha} \setminus \cup_\chi \bar{\alpha}^{\chi}, A_{\mathrm{res}}, \overline{\bar{\beta}^{\chi} \cup \bar{\alpha}^{\chi}}, \overline{A_\chi}\big) \\
\text{return} & & \mathcal{Q}', \overline{\bar{\alpha}^{\chi}_+ \bar{\alpha}^{\chi}_+}, A'_{\mathrm{res}}, \overline{\exists \bar{\alpha}^{\chi} \bar{\alpha}'^{\chi}.A'_\chi} \\
\text{else return} & & \forall(\bar{\alpha} \setminus \mathrm{FV}(\wedge_\chi A_\chi))\mathcal{Q}, \overline{\bar{\alpha}^{\chi}_+}, A_{\mathrm{res}}, \overline{\exists \bar{\alpha}^{\chi}_+.A_\chi}
\end{array}
$$

Table 7. Splitting abductive answer into invariants

Proposition 17. *Let* $(\mathcal{Q}', \overline{\bar{\alpha}^{\chi}_+}, A_{\mathrm{res}}, \overline{\exists \bar{\alpha}^{\chi}.A_\chi}) \in \mathrm{Split}\big(\mathcal{Q}, \bar{\alpha}, A, \overline{\bar{\beta}^{\chi}}\big)$. *Then:*

1. $\models A_{\mathrm{res}} \wedge_\chi A_\chi \Rightarrow A$.
2. $\models A \Rightarrow \exists \overline{\bar{\alpha}^{\chi}_+}.A_{\mathrm{res}} \wedge_\chi A_\chi$.
3. $\mathcal{Q}'.A_{\mathrm{res}}$.
4. *If* A *only restricts* $\bar{\beta}^{\chi}$ *in ways that are expressible as* $\mathcal{Q}_{<\beta_\chi} \exists \bar{\beta}^{\chi} \bar{\beta}^{\chi}_1.\varphi$ *for some* $\bar{\beta}^{\chi}_1$, *and* $\models \forall \bar{\alpha} \mathcal{Q}\big[\overline{\forall \bar{\beta}^{\chi}} := \overline{\exists \bar{\beta}^{\chi}}\big].A$ *holds, then* $\mathrm{Split}\big(\mathcal{Q}, \bar{\alpha}, A, \overline{\bar{\beta}^{\chi}}\big) \neq \varnothing$.

Furthermore, if $A = A'_{\mathrm{res}} \wedge_\chi A'_\chi$, $\mathrm{FV}(A'_\chi) \# \bar{\beta}^{\chi'}$ *for* $\chi \neq \chi'$, *and* $\models \forall \bar{\alpha} \mathcal{Q}.A'_{\mathrm{res}}$, *then for some* $(\mathcal{Q}', \overline{\bar{\alpha}^{\chi}_+}, A_{\mathrm{res}}, \overline{\exists \bar{\alpha}^{\chi}.A_\chi}) \in \mathrm{Split}\big(\mathcal{Q}, \bar{\alpha}, A, \overline{\bar{\beta}^{\chi}}\big)$, $A_\chi[\bar{\alpha}^{\chi}_+ := \bar{t}] \subset A'_\chi$ *for some* $\bar{t}$.

The potential solutions to Φ will be written as $(F_{\mathrm{res}}, \overline{\exists \bar{\alpha}^{\chi}.F_\chi})$, where F_{res}, $\overline{F_\chi}$ are conjunctions of atoms, $\bar{\alpha}^{\chi} \# \mathcal{Q}$ and $\bar{\chi}$ range over $\mathrm{PV}(\Phi)$. $(F_{\mathrm{res}}, \overline{\exists \bar{\alpha}^{\chi}.F_\chi})$ is a *solution to the inference problem* Φ when:

$\mathrm{NF}(\Phi[\overline{\chi(\tau)} := \overline{\exists \bar{\alpha}^{\chi}.F_\chi[\delta := \tau]}]) = \mathcal{Q}.\Phi_{\mathrm{PN}}$, $\forall \alpha \in \mathrm{FV}(F_\chi)$: $\alpha \in \delta\bar{\alpha}^{\chi} \vee \alpha <_{\mathcal{Q}} \beta_\chi$, $\mathcal{M} \models F_{\mathrm{res}} \Rightarrow \Phi_{\mathrm{PN}}$ and $\mathcal{M} \models \mathcal{Q}.F_{\mathrm{res}}$.

We argue that the above notion of a solution to the inference problem Φ captures the information that we want to derive, while just deciding whether thre is an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \Phi$ provides too little information.

Proposition 18. *If a solution to the inference problem* Φ *exists, then there is an interpretation of predicate variables* $\mathcal{I}$ *such that* $\mathcal{I} \models \Phi$.

Under what conditions the converse statement is true is a theoretical issue we will not pursue.

Define: $\overline{F^0_\chi} = \overline{\top}$, $\Psi_0 = \big\{\big(\top, \overline{F^0_\chi}\big)\big\}$,

$$\Psi_{k+1} = \bigcup_{(F^k_{\mathrm{res}}, \overline{\exists \bar{\alpha}^{\chi,k}.F^k_\chi}) \in \Psi_k} \Psi\big(\Phi, \overline{\exists \bar{\alpha}^{\chi,k}.F^k_\chi}\big)$$

where the operator $\Psi\big(\Phi, \overline{\exists \bar{\alpha}^{\chi,k}.F^k_\chi}\big)$ is defined as follows.

Let $\mathrm{NF}\big(\Phi\big[\overline{\chi(\tau)} := \overline{\exists \bar{\alpha}^{\chi,k}.F^k_\chi[\beta := \tau]}\big]\big) \equiv \mathcal{Q}^k.\wedge_i (D^k_i \Rightarrow C^k_i) = \mathcal{Q}^k.\Phi^k_{\mathrm{PN}} \in \mathcal{L}$. Let $\bar{\beta}^{\chi,k}$ be the variables in $\mathcal{Q}^k$ that correspond to the copy of $\bar{\alpha}^{\chi,k}$ variables from the $\chi(\beta_\chi) := \exists \bar{\alpha}^{\chi,k}.F^k_\chi[\delta := \beta_\chi]$ substitution. Now we define Ψ: table 8. Note that $\bar{\alpha}_j = \bar{\alpha}^j \cup_\chi \bar{\alpha}^{\chi}_j$.

$$
\begin{array}{rcl}
\mathcal{A} & = & \mathrm{Abd}\big(\mathcal{Q}^k\big[\overline{\forall \beta_\chi \bar{\beta}^{\chi,k}} := \overline{\exists \beta_\chi \bar{\beta}^{\chi,k}}\big], \overline{D^k_i, C^k_i}\big) \\
\mathcal{B} & = & \Big\{\Big(A^j_{\mathrm{res}}, \overline{\exists \bar{\alpha}^{\chi}_j.A^j_\chi}\Big)\Big| \\
 & & \qquad \exists \bar{\alpha}_j.A_j \in \mathcal{A} \wedge (\mathcal{Q}', \overline{\bar{\alpha}^{\chi}_+}, A_{\mathrm{res}}, \overline{\exists \bar{\alpha}^{\chi}.A_\chi}) \in \mathrm{Split}\big(\mathcal{Q}^k, \bar{\alpha}_j, A_j, \overline{\beta_\chi \bar{\beta}^{\chi,k}}\big)\Big\} \\
\Psi\big(\Phi, \overline{\bar{\alpha}^{\chi,k}.F^k_\chi}\big) & = & \Big\{\Big(A^j_{\mathrm{res}}, \overline{\exists \bar{\alpha}^{\chi,k} \bar{\alpha}^{\chi}_j.F^k_\chi \wedge A^j_\chi[\beta_\chi \bar{\beta}^{\chi,k} := \delta \bar{\alpha}^{\chi,k}]}\Big)\Big|\Big(A^j_{\mathrm{res}}, \overline{\exists \bar{\alpha}^{\chi}_j.A^j_\chi}\Big) \in \mathcal{B}\Big\}.
\end{array}
$$

Table 8. Invariants inference step

5.1 Iterating the Operator

Lemma 19. *Let* $\mathrm{NF}(\Phi[\overline{\chi(\tau)} := \overline{\exists \bar{\alpha}^{\chi}.F_\chi[\delta := \tau]}]) \equiv \mathcal{Q}.\wedge_i (D_i \Rightarrow C_i) = \mathcal{Q}.\Phi_{\mathrm{PN}} \in \mathcal{L}$.

Let $(F'_{\mathrm{res}}, \overline{\exists \bar{\alpha}^{\chi'}.F'_\chi}) \in \Psi(\Phi, \overline{\exists \bar{\alpha}^{\chi}.F_\chi})$. *Let*

$\mathcal{Q}'.\Phi'_{\mathrm{PN}} = \mathrm{NF}(\Phi[\overline{\chi^-(\beta^{\chi})} := \overline{\exists \bar{\alpha}^{\chi'}.F'_\chi[\delta := \beta^{\chi}]}; \overline{\chi^+(\tau)} := \overline{\exists \bar{\alpha}^{\chi}.F_\chi[\delta := \tau]}])$

Then $\mathcal{M} \vDash F'_{\text{res}} \Rightarrow \Phi'_{\text{PN}}$ *and* $\mathcal{M} \vDash \mathcal{Q}'.F'_{\text{res}}$.

Theorem 20. Correctness. *Let* $(F_{\text{res}}, \overline{\exists \bar{\alpha}^{\chi}.F_{\chi}})$ *be a potential solution to an inference problem* Φ. *If* $(F_{\text{res}}, \overline{\exists \bar{\alpha}^{\chi}.F_{\chi}}) \in \Psi(\Phi, \overline{\exists \bar{\alpha}^{\chi}.F_{\chi}})$, *then* $\mathcal{M}, \mathcal{I} \vDash \Phi$ *for* $\mathcal{I} = [\bar{\chi} := \overline{\exists \bar{\alpha}^{\chi}.F_{\chi}}]$.

Axiom 21. *(Interpolation property for* $\mathcal{M}$*.) We assume that for conjunctions of atoms* A *and any quantifier-free formula* Φ, $\mathcal{M} \vDash A \Rightarrow \Phi$ *implies that there is a conjunction of atoms* B *with* $\text{FV}(B) \subset \text{FV}(A) \cap \text{FV}(\Phi)$, $\mathcal{M} \vDash A \Rightarrow B$ *and* $\mathcal{M} \vDash B \Rightarrow \Phi$.

Lemma 22. *Let* $\text{NF}(\Phi[\overline{\chi(\tau)} := \overline{\exists \bar{\alpha}^{\chi}.F_{\chi}[\beta := \tau]}]) = \mathcal{Q}.\Phi_{\text{PN}}$. *Let*

$\mathcal{Q}^{\Delta}.\Phi^{\Delta}_{\text{PN}} = \text{NF}(\Phi[\overline{\chi^{-}(\beta^{\chi})} := \overline{\exists \bar{\alpha}^{\chi} \bar{\alpha}^{\chi}_{\Delta}.(\Delta_{\chi} \wedge F_{\chi})[\delta := \beta^{\chi}]}; \overline{\chi^{+}(\tau)} := \overline{\exists \bar{\alpha}^{\chi}.F_{\chi}[\delta := \tau]}])$

for variables $\overline{\bar{\alpha}^{\chi}_{\Delta}}$ *and conjunctions of atoms* $D_{\text{res}}, \overline{\Delta_{\chi}} \in \mathcal{L}$ *such that (1)* $\forall \alpha \in \text{FV}(\Delta_{\chi})$: $\alpha \in \delta \bar{\alpha}^{\chi} \bar{\alpha}^{\chi}_{\Delta} \vee \alpha <_{\mathcal{Q}} \beta_{\chi}$, $\vDash D_{\text{res}} \Rightarrow \Phi^{\Delta}_{\text{PN}}$, *(2)* $\vDash \mathcal{Q}^{\Delta}.D_{\text{res}}$ *and (3)* $\vDash \mathcal{Q}[\overline{\forall \beta_{\chi} \bar{\beta}^{\chi}} := \overline{\exists \beta_{\chi} \bar{\beta}^{\chi} \bar{\beta}^{\chi}_{\Delta}}].D_{\text{res}} \wedge_{\chi} \Delta^{\beta}_{\chi}$, *where* $\Delta^{\beta}_{\chi} = \Delta_{\chi}[\delta \bar{\alpha}^{\chi} \bar{\alpha}^{\chi}_{\Delta} := \beta_{\chi} \bar{\beta}^{\chi} \bar{\beta}^{\chi}_{\Delta}]$ *and* $\bar{\beta}^{\chi}$ *(resp.* $\bar{\beta}^{\chi} \bar{\beta}^{\chi}_{\Delta}$*) is the renaming in* $\mathcal{Q}$ *(resp.* $\mathcal{Q}^{\Delta}$*) of the negative occurrence of* $\exists \bar{\alpha}^{\chi}$ *(resp.* $\exists \bar{\alpha}^{\chi} \bar{\alpha}^{\chi}_{\Delta}$*). Then there is* $(F'_{\text{res}}, \overline{\exists \bar{\alpha}^{\chi'}.F'_{\chi}}) \in \Psi(\Phi, \overline{\exists \bar{\alpha}^{\chi}.F_{\chi}})$ *such that*

$$\vDash D_{\text{res}} \wedge_{\chi} \Delta^{\beta}_{\chi} \Rightarrow S(F'_{\text{res}} \wedge_{\chi} (F'_{\chi} \setminus F_{\chi})[\delta := \beta_{\chi}])$$

for all χ, *where* S *is a substitution for variables* $\overline{\bar{\alpha}^{\chi'}}$.

Define $\left(F^1_{\text{res}}, \overline{\exists \bar{\alpha}^{\chi}_1.F^1_{\chi}}\right) \leqslant_{\overline{\beta_{\chi}}} \left(F^2_{\text{res}}, \overline{\exists \bar{\alpha}^{\chi}_2.F^2_{\chi}}\right)$ as $\mathcal{M} \vDash F^1_{\text{res}} \wedge_{\chi} R(F^1_{\chi}[\delta := \beta_{\chi}]) \Rightarrow S(F^2_{\text{res}} \wedge_{\chi} F^2_{\chi}[\delta := \beta_{\chi}])$, for some substitution S of variables $\overline{\bar{\alpha}^{\chi}_2}$ and renaming R of variables $\overline{\bar{\alpha}^{\chi}_1}$, assuming $\bar{\alpha}^{\chi}_1 \# \bar{\alpha}^{\chi'}_1$ and $\bar{\alpha}^{\chi}_2 \# \bar{\alpha}^{\chi'}_2$ for $\chi \neq \chi'$.

Theorem 23. *Let* $(F^s_{\text{res}}, \overline{\exists \bar{\alpha}^{\chi}_s.F^s_{\chi}})$ *be any solution to the inference problem* Φ. *There is a chain* $\left(F^k_{\text{res}}, \overline{\exists \bar{\alpha}^{\chi,k}.F^k_{\chi}}\right) \in \Psi_k$, *with* $\left(F^{k+1}_{\text{res}}, \overline{\exists \bar{\alpha}^{\chi,k+1}.F^{k+1}_{\chi}}\right) \in \Psi(\Phi, \overline{\exists \bar{\alpha}^{\chi,k}.F^k_{\chi}})$, *such that for all* $k \geqslant 0$, $(F^s_{\text{res}}, \overline{\exists \bar{\alpha}^{\chi}_s.F^s_{\chi}}) \leqslant_{\overline{\beta_{\chi}}} \left(F^k_{\text{res}}, \overline{\exists \bar{\alpha}^{\chi,k}.F^k_{\chi}}\right)$ *where* $\overline{\beta_{\chi}}$ *are the universally quantified variables in* $\text{NF}(\Phi)$ *occurring in negative atoms* $\chi(\beta_{\chi})$.

6 Deriving the Postconditions

The typing problem for the language extended with existential-types constructs reduces to constraints similar to those for the original type system. The additional constructs introduce into the constraint predicate variables applied to existentially quantified variables in positive occurrences, and universally quantified variables in negative occurrences, just as the predicate variables introduced for recursive definitions. The difference is that for recursive definitions, uses are associated with positive occurrences, and introduction is associated with a negative occurrence, where for existential type, uses are associated with negative occurrences of the predicate variable, while introduction with positive occurrences. Therefore, while for recursive definitions we employed weakest stronger formulas, or greatest lower bounds (via abduction), for existential types we will find strongest weaker formulas, or least upper bounds, via *disjunction elimination*. Because the weaker formulas might be too weak, to gain completeness we will include the use sites of existential types in the abductive process. The combined solution will be computed using a two-phase scheme: at each iteration of the operator Ψ, first existential-type predicate variables will be "bootstrapped" by disjunction elimination, then the solution to all predicate variables will be updated via abduction.

Solutions to the existential-type predicate variables are found independently at each step. The intuition behind soundness is that when the fixpoint of Ψ is found, the lowest-upper-bound and greatest-lower-bound conditions on respective predicate variables mean that the resulting constraint holds. Unfortunately, with the simplest, two-phase scheme, we potentially lose semi-completeness: as more premises become available, solutions to the existential-type predicate variables monotonically increase – as they increase, parts of solutions to recursive-definition predicate variables may become obsolete. The loss of completeness might not matter in practice.

6.1 Disjunction Elimination

We assume that we have a *complete disjunction elimination algorithm* $\text{DisjElim}(\overline{D_i})$ at our disposal (where D_i are conjunctions of atoms in $\mathcal{L}$), that generates a quantified conjunction of atoms $\exists \bar{\alpha}.A \in \mathcal{F}$ (that is, A meets the solved form property), such that $\text{FV}(A) \subset \bar{\alpha} \cup \text{FV}(\overline{D_i})$. The answer $\exists \bar{\alpha}.A = \text{DisjElim}(\mathcal{Q}, \overline{D_i})$ meets the condition $\mathcal{M} \vDash \wedge_i (D_i \Rightarrow \exists \bar{\alpha}.A)$. Completeness is understood as: For every solution $\exists \bar{\alpha}_r.A_r$ with $\mathcal{M} \vDash \wedge_i (D_i \Rightarrow \exists \bar{\alpha}_r.A_r)$, $\mathcal{M} \vDash A \Rightarrow \exists \bar{\alpha}_r.A_r$, where $\exists \bar{\alpha}.A = \text{DisjElim}(\overline{D_i})$, with variables renamed so that $\bar{\alpha} \# \text{FV}(A_r)$.

6.2 Solving for Predicate Variables Introduced by Existential Types

An unpleasant thing to deal with in the constraints generated for type and invariant inference with "nominal" existential types are disjunctions used to select the existential type for elimination (i.e. use of the existential type): $(\vee_{(\varepsilon_K, \chi_K) \in \mathcal{E}} \varepsilon_K(\alpha_2) \doteq \alpha_0)$ where α_0 is the type of the expression being unpacked, and $\mathcal{E}$ are the existential types available. Because the type system statically determines which existential type is used, we assume that the disjunction can be solved by simplification of the constraints, and let the type inference fail with "existential type cannot be determined" otherwise.

With such disjunctions solved the constraints obtain the normalized form $\text{NF}(\Phi) = \mathcal{Q}. \wedge_i (D_i \Rightarrow C_i) \in \mathcal{L}^{(\chi)}$. Let us recall and update the definition of the operator Ψ to handle constraints generated for type and invariant inference with existential types:

The potential solutions to Φ will be written as $(F_{\text{res}}, \overline{\exists \bar{\alpha}^{\chi}.F_{\chi}}, \overline{\exists \bar{\alpha}^{\chi_K}.F_{\chi_K}})$, where F_{res}, $\overline{F_{\chi}}$, $\overline{F_{\chi_K}}$ are conjunctions of atoms, $\bar{\alpha}^{\chi} \bar{\alpha}^{\chi_K} \# \mathcal{Q}$ and $\text{PV}(\Phi) = \bar{\chi} \cup \overline{\chi_K}$, $\bar{\chi}$ are recursive-definition introduced predicate variables, $\overline{\chi_K}$ are existential-type introduced predicate variables. $(F_{\text{res}}, \overline{\exists \bar{\alpha}^{\chi}.F_{\chi}}, \overline{\exists \bar{\alpha}^{\chi_K}.F_{\chi_K}})$ is a *solution to the inference problem* Φ when:

$\text{NF}(\Phi[\overline{\chi(\tau)} := \overline{\exists \bar{\alpha}^{\chi}.F_{\chi}[\delta := \tau]}; \overline{\chi_K(\tau, \tau')} := \overline{\exists \bar{\alpha}^{\chi_K}.F_{\chi_K}[\delta := \tau; \delta' := \tau']}]) = \mathcal{Q}.\Phi_{\text{PN}}$,

$\forall \alpha \in \text{FV}(F_{\chi})$: $\alpha \in \delta \bar{\alpha}^{\chi} \vee \alpha <_{\mathcal{Q}} \beta_{\chi}$, $\text{FV}(F_{\chi_K}) \subset \bar{\alpha}^{\chi_K} \delta \delta'$,
$\mathcal{M} \vDash F_{\text{res}} \Rightarrow \Phi_{\text{PN}}$ and $\mathcal{M} \vDash \mathcal{Q}.F_{\text{res}}$.

We will define an operator $\Xi(\Phi') = \overline{\exists \bar{\alpha}^{\chi_K}.F_{\chi_K}}$ for $\text{PV}(\Phi') \subset \overline{\chi_K}$ later.

We define:

$$\overline{F^0_\chi} = \bar\top,\quad \Big(F^{k+1}_{\text{res}},\ \overline{\exists\bar\alpha^{\chi,k+1}.F^{k+1}_\chi},\ \overline{\exists\bar\alpha^{\chi_\kappa,k+1}.F^{k+1}_{\chi_\kappa}}\Big) \in \Psi^\exists_{k+1} = \bigcup_{(\_,\overline{\exists\bar\alpha^{\chi,k}.F^k_\chi},\_)\in\Psi^\exists_k} \Psi^\exists\big(\Phi, \overline{\exists\bar\alpha^{\chi,k}.F^k_\chi}\big)$$

Let $\Phi' = \Phi\big[\overline{\chi(\tau)} := \overline{\exists\bar\alpha^{\chi,k}.F^k_\chi[\beta:=\tau]}\big]$ (so $\text{PV}(\Phi') \subset \overline{\chi_K}$) and $\text{NF}(\Phi') \equiv \mathcal{Q}^k.\ \wedge_i (D^k_i \Rightarrow C^k_i)$. Let $\overline{\exists\bar\alpha^{\chi_\kappa}.F_{\chi_\kappa}} \in \Xi(\Phi')$ and $\overline{D^{k,e}_i, C^{k,e}_i} = \overline{D^k_i, C^k_i}\,[\overline{\chi_K(\tau,\tau')} := \overline{\exists\bar\alpha^{\chi_\kappa}.F_{\chi_\kappa}[\delta:=\tau;\delta':=\tau']}]$ where e indexes elements of $\Xi(\Phi')$. Let $\bar\beta^{\chi,k}$ be the variables in $\mathcal{Q}^k$ that correspond to the copy of $\bar\alpha^{\chi,k}$ variables from the $\chi(\beta_\chi) := \exists\bar\alpha^{\chi,k}.F^k_\chi[\delta := \beta_\chi]$ substitution. We define the updated invariants inference step in table 9 – where Φ' contains subformulas $(C^i_K)^i_K$ – the branches that introduce the existential type K – and a subformula $(C^i_K)^i_K$ is in scope of premises whose conjunction, ignoring quantifiers, forms D^i_K.

We define $\Psi^\exists$ nearly as Ψ before:

$$\begin{aligned}
\mathcal{A} &= \text{Abd}\Big(\mathcal{Q}^k\big[\overline{\forall\beta_\chi\bar\beta^{\chi,k}} := \overline{\exists\beta_\chi\bar\beta^{\chi,k}}\,\big], \overline{D^{k,e}_i, C^{k,e}_i}\Big)\\
\mathcal{B} &= \Big\{\Big(A^j_{\text{res}}, \overline{\exists\bar\alpha^\chi_j.A^j_\chi}\Big)\Big|\\
&\qquad \exists\bar\alpha_j.A_j \in \mathcal{A} \wedge (\mathcal{Q}', \overline{\bar\alpha^\chi_+}, A_{\text{res}}, \overline{\exists\bar\alpha^\chi.A_\chi}) \in \text{Split}\big(\mathcal{Q}^k, \bar\alpha_j, A_j, \overline{\beta_\chi\bar\beta^{\chi,k}}\big)\Big\}\\
\Psi^\exists_{k+1,e} &= \Big\{\Big(A^j_{\text{res}}, \overline{\exists\bar\alpha^{\chi,k}\bar\alpha^\chi_j.F^k_\chi \wedge A^j_\chi[\beta_\chi\bar\beta^{\chi,k} := \delta\bar\alpha^{\chi,k}]}\Big)\Big|\Big(A^j_{\text{res}}, \overline{\exists\bar\alpha^\chi_j.A^j_\chi}\Big) \in \mathcal{B}\Big\}.\\
\Psi^\exists\big(\Phi, \overline{\bar\alpha^{\chi,k}.F^k_\chi}\big) &= \bigcup_e \Big\{\Big(A^j_{\text{res}}, \overline{\bar\alpha^{\chi,k+1}.F^{k+1}_\chi}, \overline{\exists\bar\alpha^{\chi_\kappa}.F_{\chi_\kappa}}\Big)\Big|\Big(A^j_{\text{res}}, \overline{\bar\alpha^{\chi,k+1}.F^{k+1}_\chi}\Big) \in \Psi^\exists_{k+1,e}\Big\}
\end{aligned}$$

Now we define

$$\begin{aligned}
\Xi(\Phi') &= \overline{\Xi'(\mathcal{Q}^k, \bar\alpha^{\chi_\kappa}, F_{\chi_\kappa})}_{\chi_K\in\text{PV}^2(\Phi')} \text{ for } \exists\bar\alpha^{\chi_\kappa}.F_{\chi_\kappa} = \text{DisjElim}\big(\overline{D^i_K \wedge C^i_K \wedge \delta' \doteq \alpha^i_3}\big)\\
\Xi'(\mathcal{Q}^k, \bar\alpha^{\chi_\kappa}, F_{\chi_\kappa}) &= \exists\bar\alpha^{\chi_\kappa}\text{FV}(F_{\chi_K})\backslash\{\delta,\delta'\}.\delta \doteq \overrightarrow{\{\alpha \in \text{FV}(F_{\chi_K}) | \alpha <_{\mathcal{Q}^k} \alpha_2\}} \wedge F_{\chi_K}[\alpha_2 := \delta]\\
&\quad \text{where } \alpha_2, \alpha^i_3 \text{ are as in the subformula } (\exists\alpha^i_3.... \wedge \chi_K(\alpha_2,\alpha^i_3))^i_K
\end{aligned}$$

Table 9. Invariants and postconditions inference step

Proposition 24. Correctness. *Let* $(F_{\text{res}}, \overline{\exists\bar\alpha^\chi.F_\chi}, \overline{\exists\bar\alpha^{\chi_\kappa}.F_{\chi_\kappa}})$ *be a potential solution to an inference problem* Φ *for the language extended with existential types. Let* $\Phi' = \Phi[\overline{\chi(\tau)} := \overline{\exists\bar\alpha^\chi.F_\chi[\delta:=\tau]}]$ *and* $\overline{\exists\bar\alpha^{\chi_\kappa}.F_{\chi_\kappa}} \in \Xi(\Phi')$. *If* $(F_{\text{res}}, \overline{\exists\bar\alpha^\chi.F_\chi}, \overline{\exists\bar\alpha^{\chi_\kappa}.F_{\chi_\kappa}}) \in \Psi^\exists(\Phi, \overline{\exists\bar\alpha^\chi.F_\chi})$, *then* $\mathcal{M}, \mathcal{I} \vDash \Phi$ *for* $\mathcal{I} = [\overline{\chi} := \overline{\exists\bar\alpha^\chi.F_\chi}\,;\, \overline{\chi_K} := \overline{\exists\bar\alpha^{\chi_\kappa}.F_{\chi_\kappa}}]$.

Remark 25. We do not have even the following conservation of completeness-like result. Consider proposition *Conservative Translation* from [17]. Let ce, ce′′, C, Γ, Σ, τ, $\overline{\tau_K}$, u be as in proposition *Conservative Translation*, and let $(F_{\text{res}}, \overline{\exists\bar\alpha^{\chi'}.F'_\chi}) \in \Psi(\Phi, \overline{\exists\bar\alpha^\chi.F_\chi})$ for constraint Φ generated for C, Γ, $\Sigma \vdash$ ce′′: $u(\tau)$. Then we would like $\Psi^\exists(\Phi^\exists, \overline{\exists\bar\alpha^\chi.F_\chi}) \neq \varnothing$ for constraint $\Phi^\exists$ generated for C, $\Gamma, \Sigma \vdash$ ce: τ.

Let us compare the `ExCases` and `Clause-Abs` constraints:

$$\begin{aligned}
&[\![\Gamma, \Sigma \vdash \lambda[K]\overline{p_i.e_i} : \tau]\!]\\
=\ &\exists\alpha_1\alpha_2.(\alpha_1 \to \varepsilon_K(\alpha_2) \doteq \tau) \wedge\\
&\wedge_i([\![\Sigma \vdash p_i{\downarrow}\alpha_1]\!] \wedge\\
&\quad \forall\bar\beta'_i.D_i \Rightarrow (\exists\alpha^i_3.[\![\Gamma\Gamma'_i, \Sigma \vdash e_i : \alpha^i_3]\!] \wedge \chi_K(\alpha_2,\alpha^i_3))^i_K)
\end{aligned}$$

where $\exists\bar\beta'_i[D_i]\Gamma'_i$ is $[\![\Sigma \vdash p_i{\uparrow}\alpha_1]\!]$,
$\alpha_1\overline{\alpha^i_3}\,\bar\beta'_i \# \text{FV}(\Gamma, \Sigma, \tau, \overline{D_i})$,
$K :: \forall\alpha\gamma[\chi_K(\alpha,\gamma)].\gamma \to \varepsilon_K(\alpha) \in \Sigma$

$$\begin{aligned}
&[\![\Gamma, \Sigma \vdash \lambda\overline{p_i.e_i} : \tau]\!]\\
=\ &\exists\alpha_1\alpha_2.(\alpha_1 \to \alpha_2 \doteq \tau) \wedge_i\\
&([\![\Sigma \vdash p_i{\downarrow}\alpha_1]\!] \wedge \forall\bar\beta_i.D_i \Rightarrow [\![\Gamma\Gamma'_i, \Sigma \vdash e_i : \alpha_2]\!])
\end{aligned}$$

where $\exists\bar\beta_i[D_i]\Gamma'_i$ is $[\![\Sigma \vdash p_i{\uparrow}\alpha_1]\!]$, $\bar\beta_i \# \text{FV}(\Gamma, \Sigma, \alpha_2)$

Because Ξ computes $\text{DisjElim}\big(\overline{D^i_K \wedge [\![\Gamma\Gamma'_i, \Sigma \vdash e_i : \alpha^i_3]\!] \wedge \delta' \doteq \alpha^i_3}\big)$ which is defined as a strongest weaker formula, these differences between constraints do not spoil satisfiability for $\Phi^\exists$. But now compare the `ExLetIn` and `Abs-App` constraints:

$$\begin{aligned}
&[\![\Gamma, \Sigma \vdash \textbf{let}\ p = e_1\ \textbf{in}\ e_2 : \tau]\!]\\
=\ &\exists\alpha_0\alpha_2.(\vee_{(\varepsilon_K,\chi_K)\in\mathcal{E}}\varepsilon_K(\alpha_2) \doteq \alpha_0) \wedge [\![\Gamma, \Sigma \vdash e_1 : \alpha_0]\!] \wedge\\
&\forall\alpha_3.((\vee_{(\varepsilon_K,\chi_K)\in\mathcal{E}}(\varepsilon_K(\alpha_2) \doteq \alpha_0 \wedge \chi_K(\alpha_2,\alpha_3))) \Rightarrow\\
&\qquad ([\![\Sigma \vdash p{\downarrow}\alpha_3]\!] \wedge (\forall\bar\beta'.D \Rightarrow [\![\Gamma\Gamma', \Sigma \vdash e_2 : \tau]\!])))
\end{aligned}$$

where $\exists\bar\beta'[D]\Gamma'$ is $[\![\Sigma \vdash p{\uparrow}\alpha_3]\!]$,
$\alpha_0\alpha_2\alpha_3\bar\beta' \# \text{FV}(\Gamma, \Sigma, \tau)$,
$\mathcal{E} = \{\varepsilon_K, \chi_K | K :: \forall\alpha\gamma[\chi_K(\alpha,\gamma)].\gamma \to \varepsilon_K(\alpha) \in \Sigma\}$

$$\begin{aligned}
&[\![\Gamma, \Sigma \vdash \lambda(p.e''_2)\, e''_1 : \tau]\!]\\
=\ &\exists\alpha.[\![\Gamma, \Sigma \vdash e''_1 : \alpha]\!] \wedge\\
&[\![\Sigma \vdash p{\downarrow}\alpha]\!] \wedge \forall\bar\beta.D \Rightarrow [\![\Gamma\Gamma', \Sigma \vdash e''_2 : \tau]\!]
\end{aligned}$$

where $\exists\bar\beta[D]\Gamma'$ is $[\![\Sigma \vdash p{\uparrow}\tau_1]\!]$, $\bar\beta \# \text{FV}(\Gamma, \Sigma, \tau_2)$

Since $\chi_K(\alpha_2, \alpha_3)$ stores all the information available to reason about α_3, the constraint $\Phi^\exists$ can be unsatisfiable, due to $\alpha^i_3 \doteq \tau_K$ not being implied by some $D^i_K \wedge [\![\Gamma\Gamma'_i, \Sigma \vdash e_i : \alpha^i_3]\!]$, although clearly not being contradicted by it.

To restore completeness, we need to extend $\Psi^\exists$ into $\Psi^\exists_\forall$ that infers required facts about α_3. We describe $\Psi^\exists_\forall$ informally. The difference with $\Psi^\exists$ is that in negative positions χ_K are treated just as recursive definition predicate variables: each occurrence of $\forall\alpha_3.((\varepsilon_K(\alpha_2) \doteq \alpha_0 \wedge \chi_K(\alpha_2,\alpha_3)) \Rightarrow ...)$ is translated into $\forall\beta^j_{\chi_K}.\big(\big(\varepsilon_K(\alpha_2) \doteq \alpha_0 \wedge \big(\exists\overline{\alpha^k_{\chi_K}}.F^k_{\chi_K}\big) \wedge \chi^j_K\big(\beta^j_{\chi_K}\big)\big) \Rightarrow ...\big)$. After the solution is found, contributions from different use sites are collapsed: $\exists\overline{\alpha^{k+1}_{\chi_K}}.F^{k+1}_{\chi_K} := \exists\overline{\alpha^D_{\chi_K}}.F^D_{\chi_K} \wedge_j \exists\overline{\alpha^{k+1}_{\chi^j_K}}.F^{k+1}_{\chi^j_K}$ where $\exists\overline{\alpha^D_{\chi_K}}.F^D_{\chi_K} = \Xi(\Phi')$. The use of disjunction elimination ensures richer postconditions.

Conjecture 26. *Theorem 23 extends to the case of constraints derived for the type system extended with* `CstrIntro`, `ExCases` *and* `ExLetIn` *and the operator* $\Psi^{\exists}_{\forall}$.

7 Conclusions and Further Work

We have set out to develop an invariant inference framework around constraint based type inference for GADTs, utilizing a formulation parametric w.r.t. the domain of constraints leaving open what data properties can be expressed. For the difficult task of inference, rather than verification, of arbitrary invariants, we have given up decidability and principal types. Realizing that flexibility of invariant inference requires abstract postconditions, we have introduced implicitly generated existential types into the system. As in traditional invariant inference, we build the invariants by iteration, its each step monotonically strengthens the invariants. We are able to give completeness-like guarantees relating the derived types and invariants to what can be expressed in the type system, without limiting the full GADT-with-constraints type system in any way – on the contrary, extending its expressivity with existential types. It has turned out that the task of dividing the abductive answer into invariants of arbitrarily nested recursive definitions (and possibly, requirements on postconditions), is far from straightforward.

The first domain other than the term algebra that we will develop for is real linear arithmetic. The remaining foundational work is to devise disjunction elimination algorithms, for combining domains, for the term algebra and for linear arithmetic. Abduction algorithm for the term algebra is provided in [7], and for the linear arithmetic in [8], although further work driven by practical issues might be needed. Next we will implement the invariant inference system as described. Then, to make the system work, we will implement heuristics locating fixpoints regarding linear arithmetic constraints. Otherwise the iteration of the invariant inference operator could generate infinite progressions of uninformative constraints, where only the limit, in a closed form, captures the semantic content we seek.

Bibliography

[1] P. Cousot and R. Cousot. Automatic synthesis of optimal invariant assertions: mathematical foundations. *SIGPLAN Notices*, 12(8):1–12, aug 1977.

[2] Fritz Henglein. Type inference with polymorphic recursion. *ACM Trans. Program. Lang. Syst.*, 15(2):253–289, 1993.

[3] Kenneth W. Knowles and Cormac Flanagan. Type reconstruction for general refinement types. In *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 505–519. Springer, 2007.

[4] Chuan-kai Lin. *Practical type inference for the GADT type system*. PhD dissertation, Portland State University, Department of Computer Science, 2010.

[5] Chuan-kai Lin and Tim Sheard. Pointwise generalized algebraic data types. In *Proceedings of the 5th ACM SIGPLAN workshop on Types in language design and implementation*, TLDI '10, pages 51–62. New York, NY, USA, 2010. ACM.

[6] Michael Maher. Herbrand constraint abduction. In *LICS '05: Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science*, pages 397–406. Washington, DC, USA, 2005. IEEE Computer Society.

[7] Michael Maher and Ge Huang. On computing constraint abduction answers. In Iliano Cervesato, Helmut Veith and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 5330 of *Lecture Notes in Computer Science*, pages 421–435. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-89439-1$_3$0.

[8] MichaelJ. Maher. Abduction of linear arithmetic constraints. In Maurizio Gabbrielli and Gopal Gupta, editors, *Logic Programming*, volume 3668 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin Heidelberg, 2005.

[9] François Pottier and Yann Régis-Gianas. Stratified type inference for generalized algebraic data types. In *Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '06, pages 232–244. New York, NY, USA, 2006. ACM.

[10] Yann Regis-Gianas and Francois Pottier. A Hoare logic for call-by-value functional programs. In *Proceedings of the Ninth International Conference on Mathematics of Program Construction (MPC'08)*, volume 5133 of *Lecture Notes in Computer Science*, pages 305–335. Springer, JUL 2008.

[11] Tom Schrijvers, Simon Peyton Jones, Martin Sulzmann and Dimitrios Vytiniotis. Complete and decidable type inference for gadts. In *Proceedings of the 14th ACM SIGPLAN international conference on Functional programming*, ICFP '09, pages 341–352. New York, NY, USA, 2009. ACM.

[12] Vincent Simonet and Francois Pottier. A constraint-based approach to guarded algebraic data types. *ACM Transactions on Programming Languages and Systems*, 29(1), JAN 2007.

[13] M. Sulzmann, T. Schrijvers and P. J. Stuckey. Type inference for GADTs via Herbrand constraint abduction. Manuscript, July 2006.

[14] Hiroshi Unno and Naoki Kobayashi. Dependent type inference with interpolants. In *Proceedings of the 11th ACM SIGPLAN conference on Principles and practice of declarative programming*, PPDP '09, pages 277–288. New York, NY, USA, 2009. ACM.

[15] Łukasz Stafiniak. Joint constraint abduction problems. The International Workshop on Unification, 2011.

[16] Łukasz Stafiniak. Joint constraint abduction problems. Manuscript, 2011. Available at http://www.ii.uni.wroc.pl/~lukstafi/pubs/abduction.pdf

[17] Łukasz Stafiniak. A gadt system for invariant inference. Manuscript, 2012. Available at http://www.ii.uni.wroc.pl/~lukstafi/pubs/EGADTs.pdf

[18] Łukasz Stafiniak. Finding gadt invariants via abduction. Manuscript, 2012. Available at http://www.ii.uni.wroc.pl/~lukstafi/pubs/invariants.pdf