codegen development memo

March 4, 2022

1 Gallina

1.1 Gallina Syntax

$$
\begin{aligned}
t &= x && \text{variable}\\
&\mid c && \text{constant}\\
&\mid C && \text{constructor}\\
&\mid T && \text{type}\\
&\mid \lambda x{:}T.\, t && \text{abstraction}\\
&\mid t\, u && \text{application}\\
&\mid \texttt{let}\ x := t : T\ \texttt{in}\ u && \text{let-in}\\
&\mid \texttt{match}\ t_0\ \texttt{with}\ (C_i \Rightarrow t_i)_{i=1\ldots h}\ \texttt{end} && \text{conditional}\\
&\mid \texttt{fix}\ (f_i/k_i : T_i := t_i)_{i=1\ldots h}\ \texttt{for}\ f_j && \text{fixpoint}
\end{aligned}
$$

Note:

- u represents a term as t.
  y, z, and f represent a variable as x.
  U represents a type as T.
- We write $(\cdots((t\, u_1)\, u_2)\cdots u_n)$ as $t\, u_1 \ldots u_n$.
- k is an integer.
  k_i for fixpoint specify the decreasing argument for f_i.
- If it is unambiguous, we omit type annotations for the sake of simplicity. We also omit k_i in fixpoints if they are not used.
- We omitted the elimination predicate (`as`-`in`-`return` clause of `match`-expression). It is not used in reductions.
- `match`-branches t_i are functions that take the constructor members (constructor arguments without inductive type parameters). This formalism is taken from CIC [1].
- We omitted the detail of the types. Actual Gallina permits any Gallina term which evaluates to a type.

1.2 Gallina Conversion Rules

beta: $E[\Gamma] \vdash ((\lambda x.\, t)\; u) \triangleright t\{x/u\}$

delta-local: $\dfrac{(x := t) \in \Gamma}{E[\Gamma] \vdash x \triangleright t}$

delta-global: $\dfrac{(c := t) \in E}{E[\Gamma] \vdash c \triangleright t}$

zeta: $E[\Gamma] \vdash \texttt{let}\; x := t \;\texttt{in}\; u \triangleright u\{x/t\}$

iota-match: $\dfrac{E[\Gamma] \vdash C_j\; u_1 \ldots u_{p+m} : T \quad p \text{ is the number of parameters of the inductive type } T}{E[\Gamma] \vdash \texttt{match}\; (C_j\; u_1 \ldots u_{p+m})\; \texttt{with}\; (C_i \Rightarrow t_i)_{i=1\ldots h}\; \texttt{end} \triangleright t_j\; u_{p+1} \ldots u_{p+m}}$

iota-fix: $\dfrac{u_{k_j} = C\; u'_1 \ldots u'_m}{E[\Gamma] \vdash (\texttt{fix}\; (f_i/k_i := t_i)_{i=1\ldots h}\; \texttt{for}\; f_j)\; u_1 \ldots u_{k_j}}$

$\triangleright t_j\{f_k/\texttt{fix}\; (f_i/k_i := t_i)_{i=1\ldots h}\; \texttt{for}\; f_k\}_{k=1\ldots h}\; u_1 \ldots u_{k_j}$

eta expansion: $\dfrac{E[\Gamma] \vdash t : \forall x{:}T.\, U}{E[\Gamma] \vdash t \triangleright \lambda x{:}T.\, (t\; x)}$

Note:

- The rules shown here are reductions, except the eta expansion.
- $t\{x/u\}$ means a term in which x in term t is replaced by u. This notation is taken from the Coq reference manual [1].
- Variables cannot conflict because Coq uses de Bruijn's indexes to represent variables.
- E is a global environment which is a list of global assumptions $(c{:}T)$, global definitions $(c := t{:}T)$, and inductive definitions $(\mathsf{Ind}\,[p]\,(\Gamma_I := \Gamma_C))$.
- Γ is a local context which is a list of local assumptions $(x{:}T)$ and local definitions $(x := t{:}T)$. The local assumptions represent variables bounded by outer abstractions and fixpoints. The local definitions represent variables bounded by outer let-in.
- If it is unambiguous, we omit type annotations in these definitions for the sake of simplicity.
- Iota-match reduces `match @cons nat 1 nil with` $(\texttt{nil} \Rightarrow t_1)$ $(\texttt{cons} \Rightarrow t_2)$ `end` to t_2 `1 nil` because `list` has one parameter $(p = 1)$ and `cons` has two members $(m = 2)$.

2 CodeGen

- Convertible Transformations
  - Inlining
  - Strip Cast
  - V-Normalization
  - S-Normalization
  - Type Normalization
  - Static Argument Normalization
  - Unused let-in Deletion
  - Call Site Replacement
  - Argument Completion
  - Move Match Argument
  - Borrow Check
  - C Variable Allocation
- C Code Generation
  - C Code Generation

3 Convertible Transformations

3.1 Inlining

3.2 Strip Cast

3.3 Eta Expansion for Function Bodies

We apply eta-expansion to function bodies of top-level functions and fix-bounded functions. This makes beta-var applicable to the bodies.

3.4 V-Normalization

3.4.1 V-Reductions

$$\text{zeta-arg: } \frac{E[\Gamma] \vdash t_i : T_i \quad t_0 \text{ is not an application} \quad t_i \text{ is not a variable} \quad x_i \text{ is a fresh variable}}{E[\Gamma] \vdash t_0\, x_1 \ldots x_{i-1}\, t_i\, t_{i+1} \ldots t_n \triangleright \mathtt{let}\, x_i := t_i : T_i\, \mathtt{in}\, t_0\, x_1 \ldots x_{i-1}\, x_i\, t_{i+1} \ldots t_n}$$

$$\text{zeta-item: } \frac{E[\Gamma] \vdash t_0 : T_0 \quad t_0 \text{ is not a variable} \quad x_0 \text{ is a fresh variable}}{\begin{array}{l} E[\Gamma] \vdash \mathtt{match}\, t_0\, \mathtt{with}\, (C_i \Rightarrow t_i)_{i=1\ldots h}\, \mathtt{end} \\ \qquad \triangleright \mathtt{let}\, x_0 := t_0 : T_0\, \mathtt{in}\, \mathtt{match}\, x_0\, \mathtt{with}\, (C_i \Rightarrow t_i)_{i=1\ldots h}\, \mathtt{end} \end{array}}$$

3.4.2 V-Normal Form

V-normal form restricts Gallina terms that (1) application arguments and (2) match items to variables.

$$\begin{array}{ll} t = x \mid c \mid C \mid T \mid \lambda x{:}T.\, t \mid \mathtt{let}\, x := t : T\, \mathtt{in}\, u & \\ \quad \mid \mathtt{fix}\, (f_i/k_i : T_i := t_i)_{i=1\ldots h}\, \mathtt{for}\, f_j & \\ \quad \mid t\, x & \leftarrow (1) \\ \quad \mid \mathtt{match}\, x\, \mathtt{with}\, (C_i \Rightarrow t_i)_{i=1\ldots h}\, \mathtt{end} & \leftarrow (2) \end{array}$$

3.5 S-Normalization

$$\text{beta-var: } \frac{0 < n \quad E[\Gamma] \vdash (\lambda x.\, t)\, y_1 \ldots y_n : T \quad T \text{ is an inductive type}}{E[\Gamma] \vdash (\lambda x.\, t)\, y_1 \ldots y_n \triangleright t\{x/y_1\}\, y_2 \ldots y_n}$$

$$\text{delta-var: } \frac{(x := y) \in \Gamma}{E[\Gamma] \vdash x \triangleright y}$$

$$\text{delta-fun: } \frac{0 \le p \quad 0 < q \quad (f := t\, x_1 \ldots x_p) \in \Gamma \quad t \text{ is one of } x, c, C, \lambda x.\, u, \mathtt{fix}\, (f_i/k_i := t_i)_{i=1\ldots h}\, \mathtt{for}\, f_j}{E[\Gamma] \vdash f\, y_1 \ldots y_q \triangleright t\, x_1 \ldots x_p\, y_1 \ldots y_q}$$

zeta-flat: $E[\Gamma] \vdash \mathtt{let}\, y := (\mathtt{let}\, x := t_1\, \mathtt{in}\, t_2)\, \mathtt{in}\, t_0 \triangleright \mathtt{let}\, x := t_1\, \mathtt{in}\, (\mathtt{let}\, y := t_2\, \mathtt{in}\, t_0)$

zeta-app: $E[\Gamma] \vdash (\mathtt{let}\, x_0 := t\, \mathtt{in}\, u)\, x_1 \ldots x_n \triangleright \mathtt{let}\, x_0 := t\, \mathtt{in}\, (u\, x_1 \ldots x_n)$

$$\text{iota-match-var: } \frac{(x := C_j\, y_1 \ldots y_{p+m} : T) \in \Gamma \quad p \text{ is the number of parameters of the inductive type } T}{E[\Gamma] \vdash \mathtt{match}\, x\, \mathtt{with}\, (C_i \Rightarrow t_i)_{i=1\ldots h}\, \mathtt{end} \triangleright t_j\, y_{p+1} \ldots y_{p+m}}$$

$$\text{iota-fix-var: } \frac{\begin{array}{c} (x_{k_j} := C\, y_1 \ldots y_m) \in \Gamma \quad f'_1 \ldots f'_h \text{ are fresh variables} \\ E[\Gamma] \vdash (\mathtt{fix}\, (f_i/k_i := t_i)_{i=1\ldots h}\, \mathtt{for}\, f_j)\, x_1 \ldots x_n : T \quad T \text{ is an inductive type} \end{array}}{\begin{array}{l} E[\Gamma] \vdash (\mathtt{fix}\, (f_i/k_i := t_i)_{i=1\ldots h}\, \mathtt{for}\, f_j)\, x_1 \ldots x_n \\ \qquad \triangleright \mathtt{let}\, f'_1 := \mathtt{fix}\, (f_i/k_i := t_i)_{i=1\ldots h}\, \mathtt{for}\, f_1\, \mathtt{in}\, \ldots \\ \qquad\quad \mathtt{let}\, f'_h := \mathtt{fix}\, (f_i/k_i := t_i)_{i=1\ldots h}\, \mathtt{for}\, f_h\, \mathtt{in} \\ \qquad\quad t_j\{f_k/f'_k\}_{k=1\ldots h}\, x_1 \ldots x_n \end{array}}$$

$$\text{iota-fix-var': } \frac{\begin{array}{c} (x_{k_j} := C\, y_1 \ldots y_m) \in \Gamma \\ (f'_1 := \mathtt{fix}\, (f_i/k_i := t_i)_{i=1\ldots h}\, \mathtt{for}\, f_1) \in \Gamma \quad \ldots \quad (f'_h := \mathtt{fix}\, (f_i/k_i := t_i)_{i=1\ldots h}\, \mathtt{for}\, f_h) \in \Gamma \\ E[\Gamma] \vdash (\mathtt{fix}\, (f_i/k_i := t_i)_{i=1\ldots h}\, \mathtt{for}\, f_j)\, x_1 \ldots x_n : T \quad T \text{ is an inductive type} \end{array}}{E[\Gamma] \vdash (\mathtt{fix}\, (f_i/k_i := t_i)_{i=1\ldots h}\, \mathtt{for}\, f_j)\, x_1 \ldots x_n \triangleright t_j\{f_k/f'_k\}_{k=1\ldots h}\, x_1 \ldots x_n}$$

3.6 Type Normalization

3.7 Static Argument Normalization

3.8 Unused let-in Deletion

$$\text{zeta-del: } \frac{x \text{ does not occur in } u \quad x \text{ is not linear} \quad \mathrm{FV}(t) \text{ does not contain linear variable}}{E[\Gamma] \vdash \texttt{let}\, x := t\, \texttt{in}\, u \rhd u}$$

Note:

- $\mathrm{FV}(t)$ means the free variables of t.

3.9 Call Site Replacement

3.10 Argument Completion

Argument completion removes partial applications by applying eta expansions.

- $\mathrm{F}[\![t]\!]$ considers t to be a top-level function or a closure-generating expression. F transforms t to be a nested abstraction expression that takes all arguments. Fixpoint expressions are allowed outside or between the abstractions.
- $\mathrm{BR}[\![t]\!]_{m,q}$ transforms a branch of a match expression into a nested abstraction expression that takes constructor members. Fixpoint expressions are not allowed.
- $\mathrm{E}[\![t \,/\, x_1 \dots x_p]\!]_q$ is a term convertible with $t\, x_1 \dots x_p$ that does not contain a partial application. $\mathrm{E}[\![t \,/\, x_1 \dots x_p]\!]_q$ traverses t while tracking the arguments for t to find closure-generating expressions. The number of arguments given to t is $p + q$. The first p arguments are $x_1 \dots x_p$ and they can be the argument of beta-var redex. The last q arguments cannot be the argument of beta-var redex.

$$\mathrm{F}[\![t]\!] = \begin{cases} \lambda x.\ \mathrm{F}[\![u]\!] & t = \lambda x.\ u \\ \texttt{fix}\,(f_i := \mathrm{F}[\![t_i]\!])_{i=1\dots h}\, \texttt{for}\, f_j & t = \texttt{fix}\,(f_i := t_i)_{i=1\dots h}\, \texttt{for}\, f_j \\ \lambda x_1 \dots \lambda x_m.\ \mathrm{E}[\![t \,/\, x_1 \dots x_m]\!]_0 & \text{otherwise} \quad \text{(eta expansion)} \end{cases}$$

$$\begin{aligned} \text{where} \quad & t : T_1 \to \dots \to T_m \to T_0 \\ & T_0 \text{ is an inductive type} \\ & x_1 \dots x_m \text{ are fresh variables} \end{aligned}$$

$$\mathrm{BR}[\![t]\!]_{m,q} = \begin{cases} \mathrm{E}[\![t \,/\,]\!]_q & m = 0 \\ \lambda x.\ \mathrm{BR}[\![u]\!]_{m-1,q} & (m > 0) \wedge (t = \lambda x.\ u) \\ \lambda x_1 \dots \lambda x_m.\ \mathrm{E}[\![t \,/\, x_1 \dots x_m]\!]_q & \text{otherwise} \quad \text{(eta expansion)} \end{cases}$$

$$\text{where} \quad x_1 \dots x_m \text{ are fresh variables}$$

$$\mathrm{E}[\![t \,/\, x_1 \dots x_p]\!]_q = \begin{cases} x & (t = x) \wedge (p = q = 0) \\ x\, x_1 \dots x_p & (t = x) \wedge \neg(p = q = 0) \wedge (r = 0) \\ \mathrm{F}[\![x\, x_1 \dots x_p]\!] & (t = x) \wedge \neg(p = q = 0) \wedge (r > 0) \\ t\, x_1 \dots x_p & ((t = c) \vee (t = C)) \wedge (r = 0) \\ \mathrm{F}[\![t\, x_1 \dots x_p]\!] & ((t = c) \vee (t = C)) \wedge (r > 0) \\ \mathrm{E}[\![u\{x/x_1\} \,/\, x_2 \dots x_p]\!]_q & (t = \lambda x.\ u) \wedge (p > 0) \wedge (r = 0) \quad \text{(beta-var)} \\ (\lambda x.\ \mathrm{E}[\![u \,/\,]\!]_{p+q-1})\, x_1 \dots x_p & (t = \lambda x.\ u) \wedge ((p = 0) \wedge (r = 0)) \\ \mathrm{F}[\![t\, x_1 \dots x_p]\!] & (t = \lambda x.\ u) \wedge (r > 0) \\ \mathrm{E}[\![u \,/\, x_0\, x_1 \dots x_p]\!]_q & t = u\, x_0 \\ \texttt{let}\, x := \mathrm{E}[\![t_1 \,/\,]\!]_0\, \texttt{in}\, \mathrm{E}[\![t_2 \,/\, x_1 \dots x_p]\!]_q & t = \texttt{let}\, x := t_1\, \texttt{in}\, t_2 \quad \text{(zeta-app)} \\ \texttt{match}\, x\, \texttt{with}\, (C_i \Rightarrow \mathrm{BR}[\![t_i]\!]_{\mathrm{NM}_{C_i}, p+q})_{i=1\dots h}\, \texttt{end}\, x_1 \dots x_p & t = \texttt{match}\, x\, \texttt{with}\, (C_i \Rightarrow t_i)_{i=1\dots h}\, \texttt{end} \\ (\texttt{fix}\,(f_i := \mathrm{F}[\![t_i]\!])_{i=1\dots h}\, \texttt{for}\, f_j)\, x_1 \dots x_p & (t = \texttt{fix}\,(f_i := t_i)_{i=1\dots h}\, \texttt{for}\, f_j) \wedge (r = 0) \\ \mathrm{F}[\![t\, x_1 \dots x_p]\!] & (t = \texttt{fix}\,(f_i := t_i)_{i=1\dots h}\, \texttt{for}\, f_j) \wedge (r > 0) \end{cases}$$

where $t : T_1 \to \cdots \to T_p \to T_{p+1} \to \cdots \to T_{p+q} \to T_{p+q+1} \to \cdots \to T_{p+q+r} \to T_0$
T_0 is an inductive type

Note:

- This transformation assumes t is not dependently typed: no type terms and no dependent `match`-expressions.
- NM_C is the number of the members of the constructor C (the number of arguments without the parameters for the inductive type):
  $\mathrm{NM}_C = m$ if $C : T_1 \to \cdots \to T_p \to T_{p+1} \to \cdots \to T_{p+m} \to T_0$ and T_0 is an inductive type which has p parameters.

3.11 Move Match Argument

$$\text{match-app: } \frac{E[\Gamma] \vdash z : T}{E[\Gamma] \vdash \texttt{match}\, x\, \texttt{as}\, x'\, \texttt{in}\, I\, \_\, \ldots\, \_\, y_1 \ldots y_{\mathrm{NI}_I}\, \texttt{return}\, T \to P\, y_1 \ldots y_{\mathrm{NI}_I}\, x'}$$

$$\texttt{with}\, (C_i\, x_{i1} \ldots x_{i\,\mathrm{NM}_{C_i}} \Rightarrow t_i)_{i=1\ldots h}\, \texttt{end}\, z$$

$$\triangleright\, \texttt{match}\, x\, \texttt{as}\, x'\, \texttt{in}\, I\, \_\, \ldots\, \_\, y_1 \ldots y_{\mathrm{NI}_I}\, \texttt{return}\, P\, y_1 \ldots y_{\mathrm{NI}_I}\, x'$$

$$\texttt{with}\, (C_i\, x_{i1} \ldots x_{i\,\mathrm{NM}_{C_i}} \Rightarrow t_i\, z)_{i=1\ldots h}\, \texttt{end}$$

beta-var and zeta-app are also applied.
Note:

- match-app is not convertible
- NI_I is the number of the indexes of the inductive type I (the number of arguments without the parameters for the inductive type):
  $\mathrm{NI}_I = m$ if $I : T_1 \to \cdots \to T_p \to T_{p+1} \to \cdots \to T_{p+m} \to S$ and S is a sort.

3.12 Borrow Check

We define two judgements $E[\Gamma'] \vdash t\, T \mid B$ and $E[\Gamma'] \vdash t\, T \mid (L, B^{\mathrm{used}}, B^{\mathrm{result}})$ for borrow check. Γ' is an annotated local context. It is a list of $(x^B{:}T)$ or $(x^B := t{:}T)$. The variable x is annotated with a borrow information B. B is a set of pair of borrow type and linear variable, such as $\{(T_1, x_1), \ldots\}$. B^{used} and B^{result} are also borrow information. L is a set of linear variables. T is the type of t.

We omit $: T$ in a rule which does not use T.

The borrow information $B = \{(T_1, x_1), \ldots\}$ represents a linear variable x_i is used via borrow type T_i. $(x^{\{(T',y)\}}{:}T) \in \Gamma'$ represents x may contain a value of type T' which is a (part of) content of the linear variable y.

$E[\Gamma'] \vdash t \mid B$ means a function t may use linear variables via borrow B.

$E[\Gamma'] \vdash t \mid (L, B^{\mathrm{used}}, B^{\mathrm{result}})$ means an expression t (1) consumes linear variables L, (2) may use linear values via borrow B^{used}, (3) result value may contain linear values via borrow B^{result}.

For example, assume linear list `lseq`, borrow list `bseq` which has constructors `bnil` and `bcons`, borrow function `borrow : lseq nat` $\to$ `bseq nat`. In a code fragment `let y := borrow x in match y with bnil` $\Rightarrow$ `true | bcons h t` $\Rightarrow$ `false end` contains variables `x : lseq nat`, `y : bseq nat`, `h : nat`, and `t : bseq nat`. `y` and `t` contain a `bseq nat` value borrowed from `x`. It is represented as $\texttt{y}^{\{(\texttt{bseq nat},\texttt{x})\}}$: `bseq nat` and $\texttt{t}^{\{(\texttt{bseq nat},\texttt{x})\}}$: `bseq nat`. The type of `h` is `nat`. Since `nat` is not a borrow type, `h` lives even after `x` is consumed.

borrow-lvar: $\dfrac{(x^B{:}T) \in \Gamma' \quad x \text{ is linear}}{E[\Gamma'] \vdash x \mid (\{x\}, B, B)}$

borrow-var: $\dfrac{(x^B{:}T) \in \Gamma' \quad x \text{ is not linear}}{E[\Gamma'] \vdash x \mid (\varnothing, B, B)}$

borrow-constant: $\dfrac{c \text{ is not a borrow function}}{E[\Gamma'] \vdash c \mid (\varnothing, \varnothing, \varnothing)}$

borrow-constructor: $E[\Gamma'] \vdash C \mid (\varnothing, \varnothing, \varnothing)$

$$\text{borrow-letin: } \dfrac{\begin{array}{c} E[\Gamma'] \vdash t_1 \mid (L_1, B_1^{\text{used}}, B_1^{\text{result}}) \\ E[\Gamma' :: (x^{B_1^{\text{result}}} := t_1{:}T)] \vdash t_2 \mid (L_2, B_2^{\text{used}}, B_2^{\text{result}}) \\ L_1 \cap L_2 = \varnothing \\ x \text{ is linear} \rightarrow x \in L_2 \\ L_1 \cap B_2^{\text{used}} = \varnothing \end{array}}{E[\Gamma'] \vdash \texttt{let}\, x := t_1 : T \,\texttt{in}\, t_2 \mid (L_1 \cup L_2 - \{x\}, B_1^{\text{used}} \cup B_2^{\text{used}} - \{x\}, B_2^{\text{result}} - \{x\})}$$

$$\text{borrow-match: } \dfrac{\begin{array}{c} E[\Gamma'] \vdash t_0 \mid (L_0, B_0^{\text{used}}, B_0^{\text{result}}) \\ B_{ij} = B_0^{\text{result}} \text{ filtered with the type of } x_{ij} \\ \Gamma'_i = (x_{i1}^{B_{i1}}{:}T_{i1}) :: \cdots :: (x_{i\,\mathrm{NM}_{C_i}}^{B_{i\,\mathrm{NM}_{C_i}}}{:}T_{i\,\mathrm{NM}_{C_i}}) \\ E[\Gamma' :: \Gamma'_i] \vdash t_i \mid (L_i, B_i^{\text{used}}, B_i^{\text{result}}) \\ L_i^{\mathrm{M}} = L_i \cap \{x_{i1}, \ldots, x_{i\,\mathrm{NM}_{C_i}}\} \quad L_i^{\mathrm{F}} = L_i - \{x_{i1}, \ldots, x_{i\,\mathrm{NM}_{C_i}}\} \\ L_i^{\mathrm{M}} = \{x_{ij} | 1 \le j \le \mathrm{NM}_{C_i} \ \wedge\ x_{ij} \text{ is linear}\} \quad L_1^{\mathrm{F}} = \cdots = L_h^{\mathrm{F}} \quad L_0 \cap L_1^F = \varnothing \\ B'^{\text{used}}_i = B_1^{\text{used}} - \{x_{i1}, \ldots, x_{i\,\mathrm{NM}_{C_i}}\} \quad B'^{\text{result}}_i = B_1^{\text{result}} - \{x_{i1}, \ldots, x_{i\,\mathrm{NM}_{C_i}}\} \\ B^{\text{used}} = B'^{\text{used}}_1 \cup \cdots \cup B'^{\text{used}}_h \quad B^{\text{result}} = B'^{\text{result}}_1 \cup \cdots \cup B'^{\text{result}}_h \quad L_0 \cap B^{\text{used}} = \varnothing \end{array}}{E[\Gamma'] \vdash \texttt{match}\, t_0 \,\texttt{with}\, (C_i\, x_{i1} \ldots x_{i\,\mathrm{NM}_{C_i}} \Rightarrow t_i)_{i=1\ldots h} \,\texttt{end} \mid (L_0 \cup L_1^{\mathrm{F}}, B_0^{\text{used}} \cup B^{\text{used}}, B^{\text{result}})}$$

borrow-var-app: $\dfrac{(x_0^B{:}T') \in \Gamma' \quad \mathrm{APP}(\Gamma', B, x_1 \ldots x_n, T, L, B^{\text{used}}, B^{\text{result}})}{E[\Gamma'] \vdash x_0\, x_1 \ldots x_n : T \mid (L, B^{\text{used}}, B^{\text{result}})}$

borrow-constant-app: $\dfrac{c \text{ is not a borrow function} \quad \mathrm{APP}(\Gamma', \varnothing, x_1 \ldots x_n, T, L, B^{\text{used}}, B^{\text{result}})}{E[\Gamma'] \vdash c\, x_1 \ldots x_n : T \mid (L, B^{\text{used}}, B^{\text{result}})}$

borrow-constructor-app: $\dfrac{\mathrm{APP}(\Gamma', \varnothing, x_1 \ldots x_n, T, L, B^{\text{used}}, B^{\text{result}})}{E[\Gamma'] \vdash C\, x_1 \ldots x_n : T \mid (L, B^{\text{used}}, B^{\text{result}})}$

borrow-fix-app: $\dfrac{E[\Gamma'] \vdash \texttt{fix}\, (f_i := t_i)_{i=1\ldots h} \,\texttt{for}\, f_j \mid B \quad \mathrm{APP}(\Gamma', B, x_1 \ldots x_n, T, L, B^{\text{used}}, B^{\text{result}})}{E[\Gamma'] \vdash (\texttt{fix}\, (f_i := t_i)_{i=1\ldots h} \,\texttt{for}\, f_j)\, x_1 \ldots x_n : T \mid (L, B^{\text{used}}, B^{\text{result}})}$

$$\text{borrow-borrow: } \dfrac{\begin{array}{c} c \text{ is a borrow function} \\ E[\Gamma'] \vdash c : T^{\text{arg}} \rightarrow T^{\text{result}} \quad T^{\text{arg}} \text{ is a linear type} \quad T^{\text{result}} \text{ is a borrow type} \\ T^{\text{result}} \text{ does not contain function} \\ \{T_1, \ldots, T_n\} \text{ is a set of borrow types contained in } T^{\text{result}} \\ B = \{(T_1, x), \ldots, (T_n, x)\} \end{array}}{E[\Gamma'] \vdash c\, x \mid (\varnothing, B, B)}$$

borrow-fix-clo: $\dfrac{E[\Gamma'] \vdash \texttt{fix}\, (f_i := t_i)_{i=1\ldots h} \,\texttt{for}\, f_j \mid B}{E[\Gamma'] \vdash \texttt{fix}\, (f_i := t_i)_{i=1\ldots h} \,\texttt{for}\, f_j \mid (\varnothing, B, B)}$

borrow-abs-clo: $\dfrac{E[\Gamma'] \vdash \lambda x.\, t \mid B}{E[\Gamma'] \vdash \lambda x.\, t \mid (\varnothing, B, B)}$

$$\text{borrow-abs-fun: } \frac{\begin{array}{c} t \text{ is not an abstraction} \\ t \text{ is not a fixpoint} \\ E[\Gamma' :: (x_1^{\varnothing}{:}T_1) :: \cdots :: (x_n^{\varnothing}{:}T_n)] \vdash t : T \mid (L, B^{\text{used}}, B^{\text{result}}) \\ \{x_i | 1 \leq i \leq n \land x_i \text{ is linear}\} = L \\ B' = (B^{\text{used}} - \{x_1, \ldots x_n\}) \text{ filtered with the type of } T \end{array}}{E[\Gamma'] \vdash \lambda x_1{:}T_1.\ \ldots \lambda x_n{:}T_n.\, t \mid B'}$$

$$\text{borrow-abs-fix: } \frac{\begin{array}{c} t \text{ is a fixpoint} \\ E[\Gamma' :: (x_1^{\varnothing}{:}T_1) :: \cdots :: (x_n^{\varnothing}{:}T_n)] \vdash t \mid B \\ \forall 1 \leq i \leq n, x_i \text{ is not linear} \end{array}}{E[\Gamma'] \vdash \lambda x_1{:}T_1.\ \ldots \lambda x_n{:}T_n.\, t \mid B}$$

$$\text{borrow-fix-fun: } \frac{\begin{array}{c} E[\Gamma' :: (f_1^{\varnothing}{:}T_1) :: \cdots :: (f_h^{\varnothing}{:}T_h)] \vdash t_i \mid B_i \\ B = B_1 \cup \cdots \cup B_h \end{array}}{E[\Gamma'] \vdash \mathtt{fix}\,(f_i : T_i := t_i)_{i=1\ldots h}\ \mathtt{for}\ f_j \mid B}$$

$$\begin{aligned}
&\text{APP}(\Gamma', B, x_1 \ldots x_n, T, L, B^{\text{used}}, B^{\text{result}}) \\
&= 1 \leq n \\
&\ \land \forall 1 \leq i \leq n, \forall 1 \leq j \leq n, (i \neq j \to \neg(x_i = x_j \land x_i \text{ is linear})) \\
&\ \land L = \{x_i | 1 \leq i \leq n \land x_i \text{ is linear}\} \\
&\ \land B^{\text{used}} = B \cup \{B' | 1 \leq i \leq n \land (x_i^{B'}{:}T') \in \Gamma'\} \\
&\ \land B^{\text{result}} = B^{\text{used}} \text{ filtered with the type } T \\
&\ \land B^{\text{used}} \cap L = \varnothing
\end{aligned}$$

We mix borrow information and set of variables in set-operations. Assume $L = \{x_1, \ldots, x_n\}$ and $B = \{(T_1, y_1), \ldots, (T_m, y_m)\}$.

$$\begin{aligned}
B \cap L = L \cap B &= \{(T_i, y_i) \in B \mid 1 \leq i \leq m,\ y_i \in L\} \\
B - L &= \{(T_i, y_i) \in B \mid 1 \leq i \leq m,\ y_i \notin L\}
\end{aligned}$$

Note:

- borrow-fix-fun annotates $f_1^{\varnothing} \ldots f_n^{\varnothing}$. This is not correct because $f_1 \ldots f_n$ may refer borrowed values via free variables in $\mathtt{fix}\,(f_i : T_i := t_i)_{i=1\ldots h}\ \mathtt{for}\ f_j$. However, it is harmless because corresponding linear value cannot be consumed in the fix-term.

3.13 C Variable Allocation

$$\begin{aligned}
x &: \text{Gallina variable} \\
v &: \text{C variable} \\
V &= \textit{empty} \mid x \mapsto v \mid V; V
\end{aligned}$$

- $\text{CV}[\![t \,/\, x_1 \ldots x_n]\!]_V$ is the variable mapping of the variables declared in t.
- $x_1 \ldots x_n$ are arguments for t.
- V is the variable mapping for variables declared outside.

$$\mathrm{CV}[\![t \,/\, x_1 \ldots x_n]\!]_V = \begin{cases} \mathit{empty} & (t = x) \vee (t = c) \vee (t = C) \vee (t = T) \\ \mathrm{CV}[\![u \,/\,]\!]_{V;M}; M & (t = \lambda x.\, u) \wedge (n = 0) \quad \text{where } M = x \mapsto v \\ \mathrm{CV}[\![u \,/\, x_2 \ldots x_n]\!]_{V;M}; M & (t = \lambda x.\, u) \wedge (n > 0) \quad \text{where } M = x \mapsto V(x_1) \\ \mathrm{CV}[\![u \,/\, x_0\, x_1 \ldots x_n]\!]_V & t = u\, x_0 \\ \mathrm{CV}[\![t_1 \,/\,]\!]_V ; \mathrm{CV}[\![t_2 \,/\, x_1 \ldots x_n]\!]_{V;M}; M & t = \mathtt{let}\, x := t_1 \,\mathtt{in}\, t_2 \quad \text{where } M = x \mapsto v \\ (\,\mathrm{CV}[\![t_1 \,/\, x_1 \ldots x_n]\!]_{V;M_1}; \ldots; & t = \mathtt{match}\, x \,\mathtt{with}\, (C_i \Rightarrow \lambda y_{i1} \ldots \lambda y_{i\,\mathrm{NM}_{C_i}}.\, t_i)_{i=1 \ldots h}\, \mathtt{end} \\ \quad \mathrm{CV}[\![t_h \,/\, x_1 \ldots x_n]\!]_{V;M_h}; & \text{where } M_i = y_{i1} \mapsto v_{i1}; \ldots; y_{i\,\mathrm{NM}_{C_i}} \mapsto v_{i\,\mathrm{NM}_{C_i}} \\ \quad M_1; \ldots; M_h) & \\ \mathrm{CV}[\![t_1 \,/\,]\!]_{V;M}; \ldots; \mathrm{CV}[\![t_h \,/\,]\!]_{V;M}; M & t = \mathtt{fix}\, (f_i := t_i)_{i=1 \ldots h}\, \mathtt{for}\, f_j \\ & \text{where } M = f_1 \mapsto v_1; \ldots; f_h \mapsto v_h \end{cases}$$

where v, v_i, v_{ij} are fresh C variables

Note: We consider Gallina variables unique.

4 C Code Generation

4.1 The Gallina Subset For C Code Generation

$$E[\Gamma] \vdash_{\mathrm{b}} x \qquad E[\Gamma] \vdash_{\mathrm{b}} c \qquad E[\Gamma] \vdash_{\mathrm{b}} C$$

$$\frac{E[\Gamma] \vdash_{\mathrm{b}} t \quad E[\Gamma] \vdash x : T \quad T \text{ is a non-dependent inductive type}}{E[\Gamma] \vdash_{\mathrm{b}} t\, x}$$

$$\frac{E[\Gamma] \vdash_{\mathrm{b}} t \quad E[\Gamma :: (x := t{:}T)] \vdash_{\mathrm{b}} u \quad T \text{ is a non-dependent inductive type}}{E[\Gamma] \vdash_{\mathrm{b}} \mathtt{let}\, x := t : T \,\mathtt{in}\, u}$$

$$\frac{\begin{array}{c} E[\Gamma] \vdash x : T \quad T \text{ is a non-dependent inductive type with } p \text{ parameters: } I\, u_1 \ldots u_p \\ E[\,] \vdash C_i\, u_1 \ldots u_p : T_{i1} \to \cdots \to T_{i\,\mathrm{NM}_{C_i}} \to T \quad T_{ij} \text{ are non-dependent inductive types} \\ E[\Gamma :: (y_{i1}{:}T_{i1}) :: \cdots :: (y_{i\,\mathrm{NM}_{C_i}}{:}T_{i\,\mathrm{NM}_{C_i}})] \vdash_{\mathrm{b}} t_i \end{array}}{E[\Gamma] \vdash_{\mathrm{b}} \mathtt{match}\, x \,\mathtt{with}\, (C_i \Rightarrow \lambda y_{i1} \ldots \lambda y_{i\,\mathrm{NM}_{C_i}}.\, t_i)_{i=1 \ldots h}\, \mathtt{end}}$$

$$\frac{E[\Gamma :: (x{:}T)] \vdash_{\mathrm{b}} t \quad T \text{ is a non-dependent inductive type}}{E[\Gamma] \vdash_{\mathrm{b}} \lambda x{:}T.\, t}$$

$$\frac{E[\Gamma] \vdash_{\mathrm{f}} \mathtt{fix}\, (f_i : T_i := t_i)_{i=1 \ldots h}\, \mathtt{for}\, f_j}{E[\Gamma] \vdash_{\mathrm{b}} \mathtt{fix}\, (f_i : T_i := t_i)_{i=1 \ldots h}\, \mathtt{for}\, f_j}$$

$$\frac{E[\Gamma] \vdash t : T \quad T \text{ is a non-dependent inductive type} \quad E[\Gamma] \vdash_{\mathrm{b}} t}{E[\Gamma] \vdash_{\mathrm{f}} t}$$

$$\frac{E[\Gamma :: (x{:}T)] \vdash_{\mathrm{f}} t \quad T \text{ is a non-dependent inductive type}}{E[\Gamma] \vdash_{\mathrm{f}} \lambda x{:}T.\, t}$$

$$\frac{E[\Gamma :: (f_1{:}T_1) :: \cdots :: (f_h{:}T_h)] \vdash_{\mathrm{f}} t_i}{E[\Gamma] \vdash_{\mathrm{f}} \mathtt{fix}\, (f_i : T_i := t_i)_{i=1 \ldots h}\, \mathtt{for}\, f_j}$$

4.2 Detection of Inlinable Fixpoints

We detect inlinable fixpoints. “Inlinable fixpoint” means a fixpoint, $\mathtt{fix}\,(f_i/k_i := t_i)_{i=1\ldots h}\,\mathtt{for}\,f_j$, which all application to f_i is located at the tail positions of $f_1 \ldots f_h$. In this case, the continuation of the applications to $f_1 \ldots f_h$ in $\mathtt{let}\,x := (\mathtt{fix}\,(f_i/k_i := t_i)_{i=1\ldots h}\,\mathtt{for}\,f_j)\; x_1 \ldots x_n \,\mathtt{in}\, u$ are always $\mathtt{let}\,x := \square\,\mathtt{in}\,u$. Thus, we can translate the tail positions of $f_1 \ldots f_h$ to (1) assignments to the arguments of f_i and $\mathtt{goto}\ f_i$ for application to f_i and (2) assignment to x and $\mathtt{goto}\ u$ otherwise. This translation is equivalent to inlining a recursive function, which means generating a loop at a non-tail position.

$\mathrm{TR}[\![t]\!]_n$ is the first element of $\mathrm{RNT}[\![t]\!]_n$. $(R, N, T) = \mathrm{RNT}[\![t]\!]_n$ classify variables in t assuming that it is called with n arguments as $t\, x_1 \ldots x_n$:

R : tail-recursive fixpoint bounded functions that do not need to be real functions

N : free variables at non-tail positions of t

T : free variables at tail positions of t

"tail position" is extended to the function position of the application at a tail position.
R distinguishes fixpoint bounded functions translatable without actual functions (but with `goto`) or not.

$$\mathrm{TR}[\![t]\!]_n = R \quad \text{where} \quad (R, N, T) = \mathrm{RNT}[\![t]\!]_n$$

$$\mathrm{RNT}[\![t]\!]_n =$$
$$\begin{cases}
(\varnothing, \varnothing, \{x\}) & t = x \\
(\varnothing, \varnothing, \varnothing) & (t = c) \lor (t = C) \\
(R, N \cup \{x\}, T) & t = u\, x \quad \text{where} \quad (R, N, T) = \mathrm{RNT}[\![u]\!]_{n+1} \\
(R_1 \cup R_2, N_1 \cup T_1 \cup N_2 - \{x\}, T_2 - \{x\}) & t = \mathtt{let}\, x := t_1\, \mathtt{in}\, t_2 \\
 & \text{where} \quad (R_1, N_1, T_1) = \mathrm{RNT}[\![t_1]\!]_0 \quad (R_2, N_2, T_2) = \mathrm{RNT}[\![t_2]\!]_n \\
\left(\bigcup_{i=1}^h R_i, \bigcup_{i=1}^h N_i, \bigcup_{i=1}^h T_i\right) & t = \mathtt{match}\, x\, \mathtt{with}\, (C_i \Rightarrow t_i)_{i=1\ldots h}\, \mathtt{end} \\
 & \text{where } (R_i, N_i, T_i) = \mathrm{RNT}[\![t_i]\!]_{n+\mathrm{NM}_{C_i}} \\
(R, N - \{x\}, T - \{x\}) & (t = \lambda x.\, u) \land (n > 0) \quad \text{where} \quad (R, N, T) = \mathrm{RNT}[\![u]\!]_{n-1} \\
(R, (N \cup T) - \{x\}, \varnothing) & (t = \lambda x.\, u) \land (n = 0) \quad \text{where} \quad (R, N, T) = \mathrm{RNT}[\![u]\!]_{\mathrm{NA}_u} \quad \text{(closure)} \\
\big(\bigcup_{i=1}^h R_i \cup \{f_1, \ldots, f_h\}, & (t = \mathtt{fix}\, (f_i := t_i)_{i=1\ldots h}\, \mathtt{for}\, f_j) \\
\ \bigcup_{i=1}^h N_i - \{f_1, \ldots, f_h\}, & \land (n = \mathrm{NA}_t) \land \left(\bigcup_{i=1}^h N_i \cap \{f_1, \ldots, f_h\} = \varnothing\right) \\
\ \bigcup_{i=1}^h T_i - \{f_1, \ldots, f_h\}\big) & \text{where} \quad (R_i, N_i, T_i) = \mathrm{RNT}[\![t_i]\!]_{\mathrm{NA}_{t_i}} \\
\big(\bigcup_{i=1}^h R_i, & (t = \mathtt{fix}\, (f_i := t_i)_{i=1\ldots h}\, \mathtt{for}\, f_j) \quad \text{(real function, maybe closure)} \\
\ \bigcup_{i=1}^h (N_i \cup T_i) - \{f_1, \ldots, f_h\}, & \land \neg\left((n = \mathrm{NA}_t) \land \left(\bigcup_{i=1}^h N_i \cap \{f_1, \ldots, f_h\} = \varnothing\right)\right) \\
\ \varnothing\big) & \text{where} \quad (R_i, N_i, T_i) = \mathrm{RNT}[\![t_i]\!]_{\mathrm{NA}_{t_i}}
\end{cases}$$

Note:

- NA_t is the number of arguments of t: $\mathrm{NA}_t = m$ if $t : T_1 \to \cdots \to T_m \to T_0$ and T_0 is an inductive type.
- The variables in t are unique. Codegen uses de Bruijn's indexes for N and T; the variables renamed by Section 3.13 for R.
- x of $\mathtt{match}\, x\, \mathtt{with}\, (C_i \Rightarrow t_i)_{i=1\ldots h}\, \mathtt{end}$ is not counted because x is not a function and does not affect the final result.

4.3 Top-Level Functions Detection

If a fixpoint needs recursive call in C, we need a real C function for it. Codegen detects such fixpoints by simulating A_K and B_K in Section 4.5 and Section 4.6 to collect application of fixpoint-bounded functions.

4.4 Outer-Variable

4.5 Translation to C for a Non-Tail Position

$\mathrm{A}_K[\![t\,/\,x_1 \ldots x_n]\!]$ generates C code for $t\, x_1 \ldots x_n$ in a non-tail position. The result expression is passed to K.

$K(e) =$ "v `=` e`;`" in simple situations.

$$
\begin{array}{ll}
\mathrm{A}_K[\![x \,/\,]\!] = K(\text{“}x\text{”}) & \\
\mathrm{A}_K[\![x \,/\, x_1 \ldots x_n]\!] = \text{“passign}(\mathrm{fvars}'[\![x]\!], x_1 \ldots x_n) & (n > 0) \wedge x \text{ is bounded by a fixpoint} \wedge \\
\qquad\qquad\qquad \texttt{goto entry\_}x\texttt{;}\text{”} & x \in \mathit{TR} \\
\mathrm{A}_K[\![x \,/\, x_1 \ldots x_n]\!] = K(\text{“}x\texttt{(}y_1\texttt{,}\ldots\texttt{,}y_o\texttt{,}x_1\texttt{,}\ldots\texttt{,}x_n\texttt{)}\text{”}) & (n > 0) \wedge x \text{ is bounded by a fixpoint} \wedge \\
& x \notin \mathit{TR} \\
\mathrm{A}_K[\![c \,/\, x_1 \ldots x_n]\!] = K(\text{“}c\texttt{(}x_1\texttt{,}\ldots\texttt{,}x_n\texttt{)}\text{”}) & n \geq 0 \\
\mathrm{A}_K[\![C \,/\, x_1 \ldots x_n]\!] = K(\text{“}C\texttt{(}x_1\texttt{,}\ldots\texttt{,}x_n\texttt{)}\text{”}) & n \geq 0 \\
\mathrm{A}_K[\![t\, x_0 \,/\, x_1 \ldots x_n]\!] = \mathrm{A}_K[\![t \,/\, x_0\, x_1 \ldots x_n]\!] & \\
\mathrm{A}_K[\![\texttt{let}\ x := t_1\ \texttt{in}\ t_2 \,/\, x_1 \ldots x_n]\!] = \text{“}\,\mathrm{A}_{K'}[\![t_1 \,/\,]\!] & \text{where} \quad K'(e) = \text{“}x\ \texttt{=}\ e\texttt{;}\text{”} \\
\qquad\qquad\qquad \mathrm{A}_K[\![t_2 \,/\, x_1 \ldots x_n]\!]\text{”} & \\
\mathrm{A}_K[\![\lambda x.\, t \,/\, x_1\, x_2 \ldots x_n]\!] = \mathrm{A}_K[\![t \,/\, x_2 \ldots x_n]\!] & (x \text{ and } x_1 \text{ are mapped to the same C variable}) \\
\mathrm{A}_K[\![\texttt{match}\ x\ \texttt{with}\ (C_i \Rightarrow \lambda y_{i1} \ldots \lambda y_{i\,\mathrm{NM}_{C_i}}.\, t_i)_{i=1\ldots h}\ \texttt{end} & \text{where} \quad x : T \\
\quad /\, x_1 \ldots x_n]\!] = &
\end{array}
$$

"`switch` `(`swfunc_T`(`x`))` `{`

...

$\mathit{caselabel}_{C_i}$`:` y_{i1} `=` $\mathit{get\_member}_{C_i 1}$`(`$x$`);` `...;`
$y_{i\,\mathrm{NM}_{C_i}}$ `=` $\mathit{get\_member}_{C_i\,\mathrm{NM}_{C_i}}$`(`$x$`);`
$\mathit{linear\_dealloc}_T$`(`$x$`);`
$\mathrm{A}_K[\![t_i \,/\, x_1 \ldots x_n]\!]$
`break;`

...

`}`"

$$
\begin{array}{ll}
\mathrm{A}_K[\![\texttt{fix}\ (f_i := t_i)_{i=1\ldots h}\ \texttt{for}\ f_j \,/\, x_1 \ldots x_n]\!] = & f_j \in \mathit{TR} \\
\quad \text{“passign}(\mathrm{fvars}[\![t_j]\!], x_1 \ldots x_n) & \text{where} \\
\quad \mathrm{GENBODY}^{\mathrm{AT}}_{K'}[\![\texttt{fix}\ (f_i := t_i)_{i=1\ldots h}\ \texttt{for}\ f_j]\!] & K'(e) = \begin{cases} K(e) & K(e) \text{ contains } \texttt{goto} \\ \text{“}K(e) & \text{otherwise} \\ \ \texttt{goto exit\_}f_j\texttt{;}\text{”} & \end{cases} \\
\quad \texttt{exit\_}f_j\texttt{:}\text{”} & \\
\mathrm{A}_K[\![\texttt{fix}\ (f_i := t_i)_{i=1\ldots h}\ \texttt{for}\ f_j \,/\, x_1 \ldots x_n]\!] = & f_j \notin \mathit{TR} \\
\quad \text{“}K(f_j\texttt{(}y_1\texttt{,}\ldots\texttt{,}y_o\texttt{,}x_1\texttt{,}\ldots\texttt{,}x_n\texttt{)}) & \\
\quad \texttt{goto skip\_}f_j\texttt{;} & \\
\quad \mathrm{GENBODY}^{\mathrm{AN}}[\![\texttt{fix}\ (f_i := t_i)_{i=1\ldots h}\ \texttt{for}\ f_j]\!] & \\
\quad \texttt{skip\_}f_j\texttt{:}\text{”} &
\end{array}
$$

Note:

- "..." means a string. A string can contain characters in typewriter font and expressions starting in italic or roman font. The former is preserved as-is. The latter embeds the value of the expression (with name translation from Gallina to C).
- Gallina types, constants, and constructors have corresponding (user-configurable) C names and they are implicitly translated. Gallina variables are translated by the mapping defined in Section 3.13.
- $\mathit{TR} = \mathrm{TR}[\![t]\!]_n$ where the translating function is defined as `Definition` $c := t$ and t is an n-arguments function.
- swfunc_T, $\mathit{caselabel}_{C_i}$, and $\mathit{get\_member}_{C_i j}$ are defined by a user to translate `match`-expressions for the inductive type T.
- $\mathrm{passign}(y_1 \ldots y_n, x_1 \ldots x_n)$ is a parallel assignment. It is translated to a sequence of assignments to assign $x_1 \ldots x_n$ into $y_1 \ldots y_n$. It may require temporary variables.

- $y_1, \ldots, y_o$ are the outer variables of the fixpoint.
- We do not define $\mathrm{A}_K[\![\lambda x.\, t\,/\,]\!]$ because we do not support closures yet.
- Actual Codegen generates $\mathrm{GENBODY}^{\mathrm{AN}}[\![]\!]$ in a different position to avoid the label $\texttt{skip\_}f_j$ and $\texttt{goto}\texttt{skip\_}f_j\texttt{;}$.
- $\mathit{linear\_dealloc}_T(x)$ is the deallocation function for the linear type T. It is empty for unrestricted types.

4.6 Translation to C for a Tail Position

$\mathrm{B}_K[\![t\,/\,x_1 \ldots x_n]\!]$ generates C code for $t\,x_1 \ldots x_n$ in a tail position. The result expression is passed to K. $K(e) =$ "$\texttt{return}\ e\texttt{;}$" in simple situations.

$$
\begin{array}{ll}
\mathrm{B}_K[\![x\,/\,]\!] = K(\text{“}x\text{”}) & \\
\mathrm{B}_K[\![x\,/\,x_1 \ldots x_n]\!] = \text{“passign}(\mathrm{fvars}'[\![x]\!], x_1 \ldots x_n) & (n > 0) \wedge x \text{ is bounded by a fixpoint} \\
\qquad\qquad\qquad\qquad \texttt{goto entry\_}x\texttt{;}\text{”} & \\
\mathrm{B}_K[\![c\,/\,x_1 \ldots x_n]\!] = K(\text{“}c\texttt{(}x_1\texttt{,} \ldots \texttt{,}x_n\texttt{)}\text{”}) & n \geq 0 \\
\mathrm{B}_K[\![C\,/\,x_1 \ldots x_n]\!] = K(\text{“}C\texttt{(}x_1\texttt{,} \ldots \texttt{,}x_n\texttt{)}\text{”}) & n \geq 0 \\
\mathrm{B}_K[\![t\,x_0\,/\,x_1 \ldots x_n]\!] = \mathrm{B}_K[\![t\,/\,x_0\,x_1 \ldots x_n]\!] & \\
\mathrm{B}_K[\![\texttt{let}\ x := t_1\ \texttt{in}\ t_2\,/\,x_1 \ldots x_n]\!] = \text{“}\,\mathrm{A}_{K'}[\![t_1\,/\,]\!] & \text{where}\quad K'(e) = \text{“}x\ \texttt{=}\ e\texttt{;}\text{”} \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{B}_K[\![t_2\,/\,x_1 \ldots x_n]\!]\text{”} & \\
\mathrm{B}_K[\![\lambda x.\, t\,/\,x_1\,x_2 \ldots x_n]\!] = \mathrm{B}_K[\![t\,/\,x_2 \ldots x_n]\!] & (x \text{ and } x_1 \text{ are mapped to the same C variable}) \\
\mathrm{B}_K[\![\texttt{match}\ x\ \texttt{with}\ (C_i \Rightarrow \lambda y_{i1} \ldots \lambda y_{i\,\mathrm{NM}_{C_i}}.\, t_i)_{i=1 \ldots h}\ \texttt{end} & \text{where}\quad x : T \\
\quad /\,x_1 \ldots x_n]\!] = & \\
\quad \text{“}\texttt{switch (}\mathit{swfunc}_T\texttt{(}x\texttt{)) \{} & \\
\qquad \ldots & \\
\qquad \mathit{caselabel}_{C_i}\texttt{:}\quad y_{i1}\ \texttt{=}\ \mathit{get\_member}_{C_i 1}\texttt{(}x\texttt{); ...;} & \\
\qquad\qquad\qquad y_{i\,\mathrm{NM}_{C_i}}\ \texttt{=}\ \mathit{get\_member}_{C_i\,\mathrm{NM}_{C_i}}\texttt{(}x\texttt{);} & \\
\qquad\qquad\qquad \mathit{linear\_dealloc}_T\texttt{(}x\texttt{);} & \\
\qquad\qquad\qquad \mathrm{B}_K[\![t_i\,/\,x_1 \ldots x_n]\!] & \\
\qquad \ldots & \\
\qquad \texttt{\}}\text{”} & \\
\mathrm{B}_K[\![\texttt{fix}\ (f_i := t_i)_{i=1 \ldots h}\ \texttt{for}\ f_j\,/\,x_1 \ldots x_n]\!] = & \\
\quad \text{“passign}(\mathrm{fvars}[\![t_j]\!], x_1 \ldots x_n) & \\
\quad \mathrm{GENBODY}^{\mathrm{B}}_K[\![\texttt{fix}\ (f_i := t_i)_{i=1 \ldots h}\ \texttt{for}\ f_j]\!]\text{”} &
\end{array}
$$

Note:

- We do not define $\mathrm{B}_K[\![\lambda x.\, t\,/\,]\!]$ because a tail position cannot be a function after the argument completion.

4.7 Auxiliary Functions for Translation to C

$$
\mathrm{fvars}[\![t]\!] = \begin{cases} \text{“}x;\ \mathrm{fvars}[\![u]\!]\text{”} & t = \lambda x.\, u \\ \mathrm{fvars}[\![t_j]\!] & t = \texttt{fix}\ (f_i := t_i)_{i=1 \ldots h}\ \texttt{for}\ f_j \\ \text{“”} & \text{otherwise} \end{cases}
$$

$$
\mathrm{fvars}'[\![f_i]\!] = \mathrm{fvars}[\![t_i]\!] \quad \text{for functions bounded by } \texttt{fix}\ (f_i := t_i)_{i=1 \ldots h}\ \texttt{for}\ f_j
$$

$$
\mathrm{GENBODY}^{\mathrm{AT}}_K[\![t]\!] = \begin{cases} \mathrm{GENBODY}^{\mathrm{AT}}_K[\![u]\!] & t = \lambda x.\, u \\ \text{“}\texttt{entry\_}f_i\texttt{:}\ \ \mathrm{GENBODY}^{\mathrm{AT}}_K[\![t_i]\!]\text{”} & t = \texttt{fix}\ (f_i := t_i)_{i=1 \ldots h}\ \texttt{for}\ f_j \\ \quad \text{for } i = j, 1, \ldots, (j-1), (j+1), \ldots, h & \\ \mathrm{A}_K[\![t\,/\,]\!] & \text{otherwise} \end{cases}
$$

$$\text{GENBODY}^{\text{AN}}[\![t]\!] = \begin{cases} \text{GENBODY}^{\text{AN}}[\![u]\!] & t = \lambda x.\, u \\ \text{“}\texttt{entry\_}f_i\texttt{:}\ \text{GENBODY}^{\text{AN}}[\![t_i]\!]\text{”} & t = \texttt{fix}\,(f_i := t_i)_{i=1\ldots h}\ \texttt{for}\ f_j \\ \quad \text{for } i = 1, \ldots, h & \\ \text{B}_K[\![t\,/\,]\!] & \text{otherwise} \\ & \text{where} \\ & \quad t : T \\ & \quad K(e) = \text{“}\texttt{*(}T\texttt{*)ret = }e\texttt{; return;}\text{”} \end{cases}$$

$$\text{GENBODY}^{\text{B}}_K[\![t]\!] = \begin{cases} \text{GENBODY}^{\text{B}}_K[\![u]\!] & t = \lambda x.\, u \\ \text{“}\texttt{entry\_}f_i\texttt{:}\ \text{GENBODY}^{\text{B}}_K[\![t_i]\!]\text{”} & t = \texttt{fix}\,(f_i := t_i)_{i=1\ldots h}\ \texttt{for}\ f_j \\ \quad \text{for } i = j, 1, \ldots, (j-1), (j+1), \ldots, h & \\ \text{B}_K[\![t\,/\,]\!] & \text{otherwise} \end{cases}$$

Note:

- fvars and fvars′ returns a list of variables: $x_1; \ldots; x_n;$. For simplicity, we omit “;” if not ambiguous.
- “$g(i)$” for $i = j_1, \ldots, j_n$ means “$g(j_1)\ \ldots\ g(j_n)$”.

4.8 Translation for a Top-Level Function which is Translated to Multiple C Functions

$\text{GENFUN}^{\text{M}}[\![c]\!]$ translates the function (constant) c with one or more auxiliary functions. We assume c is defined as `Definition` $c := t$. The auxiliary functions $f_1 \ldots f_n$ are fixpoint bounded functions in t which are invoked as functions. We assume the types of them:

$$\begin{aligned} c &: T_{01} \to \cdots \to T_{0m_0} \to T_{00} \\ f_i &: T_{i1} \to \cdots \to T_{im_i} \to T_{i0} & i = 1 \ldots n \end{aligned}$$

where $\quad T_{i0}$ are inductive types $(i = 0 \ldots n)$

The formal arguments of c are $x_{01} \ldots x_{0m_0} = \text{fvars}[\![t]\!]$ and the formal arguments of f_i are $x_{i1} \ldots x_{im_i} = \text{fvars}'[\![f_i]\!]$.

f_i invocation in C needs extra arguments, $y_{i1}{:}U_{i1} \ldots y_{io_i}{:}U_{io_i}$, addition to the actual arguments in Gallina application because the free variables of the fixpoint should also be passed. If the free variables contain a function bounded by an outer fixpoint, the function itself is not passed but the free variables of the outer fixpoint are also passed. We iterate it until no fixpoint functions.

$$\text{GENFUN}^{\text{M}}[\![c]\!] = \text{“enum\_entries}[\![c]\!]\ \text{arg\_structdefs}[\![c]\!]\ \text{forward\_decl}[\![c]\!]\ \text{entry\_functions}[\![c]\!]\ \text{body\_function}[\![c]\!]\text{”}$$

$$\text{enum\_entries}[\![c]\!] = \text{“}\texttt{enum enum\_func\_}c\,\texttt{\{func\_}c\texttt{,func\_}f_1\texttt{,} \ldots \texttt{,func\_}f_n\texttt{\};}\text{”}$$

$$\text{arg\_structdefs}[\![c]\!] = \text{“main\_structdef}[\![c]\!]\ \text{aux\_structdef}[\![c]\!]_1 \ldots \text{aux\_structdef}[\![c]\!]_n\text{”}$$

$$\text{main\_structdef}[\![c]\!] = \text{“}\texttt{struct arg\_}c\,\texttt{\{}\,T_{01}\,\texttt{arg1;} \ldots \texttt{;}\,T_{0m_0}\,\texttt{arg}m_0\texttt{; \};}\text{”}$$

$$\text{aux\_structdef}[\![c]\!]_i = \text{“}\texttt{struct arg\_}f_i\,\texttt{\{}\,U_{i1}\,\texttt{outer1;} \ldots \texttt{;}\,U_{io_i}\,\texttt{outer}o_i\texttt{;}\,T_{i1}\,\texttt{arg1;} \ldots \texttt{;}\,T_{im_i}\,\texttt{arg}m_i\texttt{; \};}\text{”}$$

$$\text{forward\_decl}[\![c]\!] = \text{“}\texttt{static void body\_function\_}c\texttt{(enum enum\_func\_}c\,\texttt{g,void *arg,void *ret);}\text{”}$$

$$\text{entry\_functions}[\![c]\!] = \text{“main\_function}[\![c]\!]\ \text{aux\_function}[\![c]\!]_1 \ldots \text{aux\_function}[\![c]\!]_n\text{”}$$

$$\begin{aligned} \text{main\_function}[\![c]\!] = \text{“}&\texttt{static}\ T_{00}\ c\texttt{(}T_{01}\ x_{01}\texttt{,} \ldots \texttt{,}\ T_{0m_0}\ x_{0m_0}\texttt{) \{} \\ &\quad \texttt{struct arg\_}c\ \texttt{arg = \{}x_{01}\texttt{,} \ldots \texttt{,}\ x_{0m_0}\texttt{\};}\ T_{00}\ \texttt{ret;} \\ &\quad \texttt{body\_function\_}c\texttt{(func\_}c\texttt{,\&arg,\&ret);return ret;} \\ &\texttt{\}}\text{”} \end{aligned}$$

$$\mathrm{aux\_function}[\![c]\!]_i = \text{“}\mathtt{static}\ T_{i0}\ f_i\mathtt{(}U_{i1}\ y_{i1}\mathtt{,}\ \ldots\ \mathtt{,}U_{io_i}\ y_{io_i}\mathtt{,}T_{i1}\ x_{i1}\mathtt{,}\ \ldots\ \mathtt{,}T_{im_i}\ x_{im_i}\mathtt{)\ \{}$$

$$\quad \mathtt{struct\ arg\_}f_i\ \mathtt{arg = \{}y_{i1}\mathtt{,}\ \ldots\ \mathtt{,}y_{io_i}\mathtt{,}x_{i1}\mathtt{,}\ \ldots\ \mathtt{,}x_{im_i}\mathtt{\};}\ T_{i0}\ \mathtt{ret;}$$

$$\quad \mathtt{body\_function\_}c\mathtt{(func\_}f_i\mathtt{,\&arg,\&ret);\ return\ ret;}$$

$$\mathtt{\}}\text{”}$$

$$\mathrm{body\_function}[\![c]\!] = \text{“}\mathtt{static\ void\ body\_function\_}c\mathtt{(enum\ enum\_func\_}c\ \mathtt{g,void\ *arg,void\ *ret)\ \{}$$

$$\quad \mathit{decls}$$

$$\quad \mathtt{switch\ (g)\ \{}\ \mathrm{aux\_case}[\![c]\!]_1 \ldots \mathrm{aux\_case}[\![c]\!]_n\ \mathrm{main\_case}[\![c]\!]\ \mathtt{\}}$$

$$\quad \mathrm{GENBODY}^{\mathrm{B}}_K[\![t]\!]$$

$$\mathtt{\}}\text{”}$$

$$\mathrm{aux\_case}[\![c]\!]_i = \text{“}\mathtt{case\ func\_}f_i\mathtt{:}$$

$$\quad y_{i1}\ \mathtt{= ((struct\ arg\_}f_i\ \mathtt{*)arg)\text{->}outer1;}\ \ldots\mathtt{;}\ y_{io_i}\ \mathtt{= ((struct\ arg\_}f_i\ \mathtt{*)arg)\text{->}outer}o_i\mathtt{;}$$

$$\quad x_{i1}\ \mathtt{= ((struct\ arg\_}f_i\ \mathtt{*)arg)\text{->}arg1;}\ \ldots\mathtt{;}\ x_{im_i}\ \mathtt{= ((struct\ arg\_}f_i\ \mathtt{*)arg)\text{->}arg}m_i\mathtt{;}$$

$$\quad \mathtt{goto\ entry\_}f_i\mathtt{;}\text{”}$$

$$\mathrm{main\_case}[\![c]\!] = \text{“}\mathtt{default:;}$$

$$\quad x_{01}\ \mathtt{= ((struct\ arg\_}c\ \mathtt{*)arg)\text{->}arg1;}\ \ldots\mathtt{;}\ x_{0m_0}\ \mathtt{= ((struct\ arg\_}c\ \mathtt{*)arg)\text{->}arg}m_0\mathtt{;}\text{”}$$

where *decls* is local variable declarations for variables used in $\mathrm{GENBODY}^{\mathrm{B}}_K[\![t]\!]$.
$K(e) = \text{“}\mathtt{*(}T_{00}\mathtt{*)ret =}\ e\mathtt{;return;}\text{”}$

4.9 Translation for a Top-Level Function which is Translated to a Single C Function

$\mathrm{GENFUN}^{\mathrm{S}}[\![c]\!]$ translates the function (constant) c to a single C function.

$$\mathrm{GENFUN}^{\mathrm{S}}[\![c]\!] = \text{“}\mathtt{static}\ T_0\ c\mathtt{(}\ \mathrm{fargs}'[\![t]\!]\mathtt{)\ \{}\ \mathit{decls}\ \ \mathrm{GENBODY}^{\mathrm{B}}_K[\![t]\!]\ \mathtt{\}}\text{”}$$

where c is defined as `Definition` $c : T_1 \to \cdots \to T_n \to T_0 := t$.
T_0 is an inductive type
decls is local variable declarations for variables used in $\mathrm{GENBODY}^{\mathrm{B}}_K[\![t]\!]$ excluding $\mathrm{fargs}[\![t]\!]$.
$K(e) = \text{“}\mathtt{return}\ e\mathtt{;}\text{”}$

$$\mathrm{fargs}[\![t]\!] = \begin{cases} \text{“}T\ x\mathtt{,}\ \ \mathrm{fargs}[\![u]\!]\text{”} & t = \lambda x{:}T.\ u \\ \mathrm{fargs}[\![t_j]\!] & t = \mathtt{fix}\ (f_i := t_i)_{i=1\ldots h}\ \mathtt{for}\ f_j \\ \text{“”} & \text{otherwise} \end{cases}$$

$\mathrm{fargs}'[\![t]\!] = \mathrm{fargs}[\![t]\!]$ without the trailing comma

4.10 Translation for Top-Level Function

$$\mathrm{GENFUN}[\![c]\!] = \begin{cases} \mathrm{GENFUN}^{\mathrm{M}}[\![c]\!] & t \text{ needs multiple functions} \\ \mathrm{GENFUN}^{\mathrm{S}}[\![c]\!] & \text{otherwise} \end{cases}$$

where c is defined as `Definition` $c := t$.

References

[1] The Coq Development Team. The coq reference manual: Release 8.12.0. 2020.