A Typed language for Interlocking Logic Verification

Nobody,[1*] Somebody,[1] Anybody[1]

[1]SOMETHING CONSTRUCTIVE CO., LTD. Tokyo Japan
[*]To whom correspondence should be addressed; E-mail: noname@nowhere.com.

Recently progressive Artificial Inteligence technology brings drastically distribution to many of industorial fields including railway and signalling engineering. This paper proposes one of the way to make use the type theory based A.I. technology on railway signalling engineering domain.

1 Motivation

Interlocking equipment acts as the vital role of train traffic safety. And its internal logic is ordinary described in the form of "wired logic notation", for the fact that interlocking equipment is constructed of relays. Nowadays we have developped electronic ones, called as electronic interlocking equipment, which consists of electronic components including microcomputers, and its interlocking logic is implemented in computer program instead. However electronic interlocking equipment isn't seem to be well accepted in railway enginerring domain. In fact most of railway signaling engineers aren't still used to make use microcomputer based programming to implement interlocking logic. In a sense, it's adequate way to use ordinary wired logic notation to implement interlocking logic on electronic interlocking equipment, accounting on the fact that the engineers have plenty experience in design of traditional relay-based interlocking equipment.

This research tries to find the methods to obtain modern environment for designing, verification and debugging in interlocking equipment development process, and increase our productivity.

2 Introduction

In this paper, we suggest the type theory based matching system suitable for interlocking logic structures. Our matching system brings a capability of pattern recongnition on interlocking logic structures, which encourges us to realize automated verification its logical consistency and correctness.

3 Basic Logic

In this section, we present our basic logic. As the first step, we define the terms, patterns and matching rules to combine them, on our matching system.

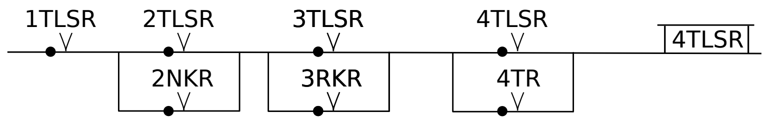
Definition 3.1 (terms) *We define the terms t range over $\mathbb{T}$ as follow.*

$$\begin{aligned} t \overset{\text{def}}{:=} \ & t_0 \ \textit{s.t.}\ (t_0, \mathcal{R}_0) \in \Gamma_{gnd} \mid t_1 \ \textit{s.t.}\ (t_1, \mathcal{R}_1) \in \Gamma_{def} \mid t_1 \wedge t_2 \mid t_1 \vee t_2 \mid \\ & \{\} \mid \{t_1 \cdots t_n\} \mid [t_1 \cdots t_n] \mid \langle t_1 \cdots t_n \rangle \mid \circ \mid {t_1}^{?} \mid {t_1}^{\leftarrow} \mid {t_1}^{\rightarrow} \end{aligned}$$

We sometime call the terms $t \in \mathbb{T}$ defined above as canonical terms in certain situation.

We can express arbitrary logics based on the above definition of terms, illustrated in Figure 5 as one of examples of terms.

Then we define some predicators and metrics on $\mathbb{T}$ as follows.



$$t_{TLSR_{4T}} = (TLSR_{1T} \vee \circ) \wedge \Big(\big\{(TLSR_{2T} \vee NKR_{\#2}{}^{?}), (TLSR_{3T} \vee RKR_{\#3}{}^{?})\big\} \wedge (TLSR_{4T} \vee TR_4)\Big)$$

Figure 1: interlocking logic example 1

Definition 3.2 (grand terms) *We define the set of grand terms of a given term $t \in \mathbb{T}$ as follows.*

$$gnd(t) := \begin{cases} \{t_0\} \quad \text{if } t = t_0 \text{ s.t. } (t_0, \mathcal{R}_0) \in \Gamma_{gnd} \\ \{t_1\} \quad \text{if } t = t_1 \text{ s.t. } (t_1, \mathcal{R}_1) \in \Gamma_{def} \\ gnd(t_1) \cup gnd(t_2) \quad \text{if } t = t_1 \wedge t_2 \text{ or } t_1 \vee t_2 \\ \{\{\}\} \quad \text{if } t = \{\} \\ \bigcup_{j=1}^{n} gnd(t_j) \quad \text{if } t = \{t_1 \cdots t_n\} \text{ or } [t_1 \cdots t_n] \text{ or } \langle t_1 \cdots t_n \rangle \text{ s.t. } n \geq 1 \\ \{\circ\} \quad \text{if } t = \circ \\ gnd(t_1) \quad \text{if } t = t_1^{?} \text{ or } t = t_1^{\leftarrow} \text{ or } t = t_1^{\rightarrow} \end{cases}$$

Definition 3.3 (subterms) *We define the set of subterms of a given term $t \in \mathbb{T}$ as follows.*

$$sub(t) \quad := \quad \begin{cases} \{t\} & \text{if } \begin{cases} t = t_0 \text{ s,t. } (t_0, \mathcal{R}_0) \in \Gamma_{gnd} \\ t = t_1 \text{ s.t. } (t_1, \mathcal{R}_1) \in \Gamma_{def} \\ t = \{\} \\ t = \circ \end{cases} \\ \{t\} \cup \big(sub(t_1) \cup sub(t_2)\big) & \text{if } t = t_1 \wedge t_2,\ t_1 \vee t_2 \\ \{t\} \cup \big(\bigcup_{j=1}^{n} sub(t_j)\big) & \text{if } t = \{t_1 \cdots t_n\},\ [t_1 \cdots t_n],\ \langle t_1 \cdots t_n \rangle \\ \{t_1{}^{?}\} \cup sub(t_1) & \text{if } t = t_1{}^{?} \\ \{t_1{}^{\leftarrow}\} \cup sub(t_1) & \text{if } t = t_1{}^{\leftarrow} \\ \{t_1{}^{\rightarrow}\} \cup sub(t_1) & \text{if } t = t_1{}^{\rightarrow} \end{cases}$$

Definition 3.4 (term size) *We define size of a given term $t \in \mathbb{T}$ as follow.*

$$size(t) \quad := \quad \begin{cases} 1 & \text{if } \begin{cases} t = t_0 \text{ s,t. } (t_0, \mathcal{R}_0) \in \Gamma_{gnd} \\ t = t_1 \text{ s.t. } (t_1, \mathcal{R}_1) \in \Gamma_{def} \\ t = \{\} \\ t = \circ \end{cases} \\ \big(size(t_1) + size(t_2)\big) + 1 & \text{if } t = t_1 \wedge t_2,\ t_1 \vee t_2 \\ \big(\sum_{j=1}^{n} size(t_j)\big) + 1 & \text{if } t = \{t_1 \cdots t_n\},\ [t_1 \cdots t_n],\ \langle t_1 \cdots t_n \rangle \\ size(t_1) + 1 & \text{if } \begin{cases} t = t_1{}^{?} \\ t = t_1{}^{\leftarrow} \\ t = t_1{}^{\rightarrow} \end{cases} \end{cases}$$

Property 3.1

$$\mathit{sub}(t') \subseteq \mathit{sub}(t) \textit{ where } t' \in \mathit{sub}(t)$$

Proof. *Proof is simple induction on sturcture of t. Here we show the proof only for the most significant case.*

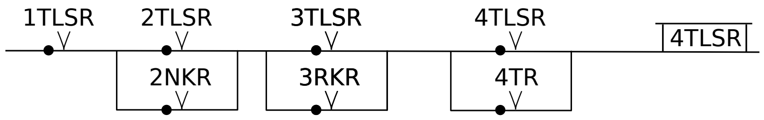
Case $\{t_1, \cdots t_n\}$: We suppose $\mathit{sub}\big(\{t_1, \cdots t_n\}\big) = \big\{\{t_1, \cdots t_n\}\big\} \cup \big(\bigcup_{j=1}^{n} \mathit{sub}(t_j)\big)$ from def. of sub. For the case of t' comming from $\mathit{sub}(t_j)$, i.e. $t' \in \mathit{sub}(t_j) \subseteq \mathit{sub}\big(\{t_1, \cdots t_n\}\big)$, we say $\mathit{sub}(t') \subseteq \mathit{sub}(t_j)$ from induction hypothesis. □

Definition 3.5 (patterns) *We define the patterns $\mathcal{R}$ range over $\mathbb{R}$ as follow.*

$$\mathcal{R} \overset{\text{def}}{:=} \mathcal{R}_0 \textit{ s.t. } (t_0, \mathcal{R}_0) \in \Gamma_{gnd} \mid \mathcal{R}_1 \textit{ s.t. } (t_1, \mathcal{R}_1) \in \Gamma_{def} \mid \mathcal{R}_1 \wedge \mathcal{R}_2 \mid \mathcal{R}_1 \vee \mathcal{R}_2 \mid$$
$$\mathcal{R}_1{}^{*} \mid \mathcal{R}_1{}^{+} \mid \mathcal{R}_1{}^{\downarrow} \mid \mathcal{R}_1{}^{?} \mid (\mathcal{R}_1 | \mathcal{R}_2)$$

Intuitively we regard patterns as descripter of the set of specific terms, i.e. $\mathcal{R}_{TLSR}$ has $TLSR_{2T}, TLSR_{3T}, TLSR_{4T}, \cdots$ and $(\mathcal{R}_{NKR}|\mathcal{R}_{RKR})$ does $NKR_{\#2}, NKR_{\#3}, \cdots$, and $\mathcal{R}_{TLSR} \vee (\mathcal{R}_{NKR}|\mathcal{R}_{RKR})$ is $TLSR_{2T} \vee NKR_{\#2}$ so forth.

We also can describe arbitrary patterns to express any of terms belonging to them. Follwing is one of example of pattern matching comming from Figure 5.



$$(TLSR_{1T} \vee \circ) \wedge \Big(\big\{(TLSR_{2T} \vee NKR_{\#2}{}^{?}), (TLSR_{3T} \vee RKR_{\#3}{}^{?})\big\} \wedge (TLSR_{4T} \vee TR_4)\Big) : \mathcal{R}_{TLSR_{4T}}$$

$$\textit{where } \mathcal{R}_{TLSR_{4T}} = \big(TLSR \vee (NKR|RKR)^{?}\big) \wedge \Big(\big(TLSR \vee (NKR|RKR)^{?}\big)^{*} \wedge (TLSR \vee TR)\Big)$$

Figure 2: pattern of interlocking logic example 1

Here we wrote $\mathcal{R}_{XXX}$ as XXX in shorthand, e.g. $TLSR$ as $\mathcal{R}_{TLSR}$. And we write $t : \mathcal{R}$ with the intention of that t matches the pattern of $\mathcal{R}$, therefore $t_{TLSR_{4T}} : \mathcal{R}_{TLSR_{4T}}$ means that $t_{TLSR_{4T}}$

matches the pattern of $\mathcal{R}_{TLSR_{4T}}$ where $t_{TLSR_{4T}} = (TLSR_{1T} \vee \circ) \wedge \Big(\big\{ (TLSR_{2T} \vee NKR_{\#2}{}^{?}), (TLSR_{3T} \vee RKR_{\#3}{}^{?}) \big\} \wedge (TLSR_{4T} \vee TR_4) \Big)$ as Figure 2.

Consequently, we define the entity holding the correspondence between term $t \in \mathbb{T}$ and patterns $\mathcal{R} \in \mathbb{R}$, as binding and some related properties.

Definition 3.6 (binding) *We define a set of pairs of $(t, \mathcal{R})$ where $t \in \mathbb{T}$ and $\mathcal{R} \in \mathbb{R}$, i.e. $\big\{(t, \mathcal{R}) \mid t \in \mathbb{T},\ \mathcal{R} \in \mathbb{R}\big\}$, as binding ranging over Γ.*

Definition 3.7 (binding domain) *We define a set of terms t comming from Γ s.t. $\big\{t \mid (t, \mathcal{R}) \in \Gamma\big\}$, as $\mathcal{D}om(\Gamma)$.*

$$\mathcal{D}om(\Gamma) \overset{\text{def}}{:=} \big\{t \mid (t, \mathcal{R}) \in \Gamma\big\}$$

We easily find some properties on $\mathcal{D}om(\Gamma)$ as follows.

Property 3.2

$$\mathcal{D}om(\Gamma_1 \cup \Gamma_2) = \mathcal{D}om(\Gamma_1) \cup \mathcal{D}om(\Gamma_2)$$

Proof. *Proof is dirctrly from def. of $\mathcal{D}om$.*

$\mathcal{D}om(\Gamma_1 \cup \Gamma_2) \subseteq \mathcal{D}om(\Gamma_1) \cup \mathcal{D}om(\Gamma_2)$: From def. of $\mathcal{D}om$, we suppose that $t \in \mathcal{D}om(\Gamma_1 \cup \Gamma_2) = \big\{t' \mid (t', \mathcal{R}') \in (\Gamma_1 \cup \Gamma_2)\big\}$, then t is t'_1 s.t. $(t'_1, \mathcal{R}'_1) \in \Gamma_1$ or t'_2 s.t. $(t'_2, \mathcal{R}'_2) \in \Gamma_2$, which means $t \in \mathcal{D}om(\Gamma_1)$ or $t \in \mathcal{D}om(\Gamma_2)$ from def. of $\mathcal{D}om$. Hence $t \in \big(\mathcal{D}om(\Gamma_1) \cup \mathcal{D}om(\Gamma_2)\big)$.

$\mathcal{D}om(\Gamma_1 \cup \Gamma_2) \supseteq \mathcal{D}om(\Gamma_1) \cup \mathcal{D}om(\Gamma_2)$: Now we suppose $t \in \big(\mathcal{D}om(\Gamma_1) \cup \mathcal{D}om(\Gamma_2)\big)$. Then from def. of $\mathcal{D}om$, t belongs to $\Big(\big\{t'_1 \mid (t'_1, \mathcal{R}'_1) \in \Gamma_1\big\} \cup \big\{t'_2 \mid (t'_2, \mathcal{R}'_2) \in \Gamma_2\big\}\Big)$, which implies that t is in $\big\{t'_1 \mid (t'_1, \mathcal{R}'_1) \in \Gamma_1\big\}$ or $\big\{t'_2 \mid (t'_2, \mathcal{R}'_2) \in \Gamma_2\big\}$. Here we assume $t \in \big\{t'_1 \mid (t'_1, \mathcal{R}'_1) \in \Gamma_1\big\} \subseteq \big\{t'' \mid (t'', \mathcal{R}'') \in (\Gamma_1 \cup \Gamma_2)\big\} = \mathcal{D}om(\Gamma_1 \cup \Gamma_2)$, and the counterpart case is also. Hence we can conclude that t is also in $\big\{t' \mid (t', \mathcal{R}') \in (\Gamma_1 \cup \Gamma_2)\big\} = \mathcal{D}om(\Gamma_1 \cup \Gamma_2)$, from def. of $\mathcal{D}om$. $\square$

Property 3.3

$$\mathcal{Dom}\left(\bigcup_{j=1}^{n}\Gamma_j\right) = \bigcup_{j=1}^{n}\mathcal{Dom}(\Gamma_j)$$

Proof. *Proof is induction on n, as num. of Γ_j s.t. $\mathcal{Dom}(\Gamma_j)$.*

B.C.): $n = 1$: Trivial. $n = 2$: From Property 3.2, $\mathcal{Dom}(\Gamma_1 \cup \Gamma_2) = \mathcal{Dom}(\Gamma_1) \cup \mathcal{Dom}(\Gamma_2)$.

I.S.): $n \geq 3$: $\mathcal{Dom}\big(\bigcup_{j=1}^{n}\Gamma_j\big) = \mathcal{Dom}\big(\Gamma_1 \cup \bigcup_{j=2}^{n}\Gamma_j\big) = \mathcal{Dom}(\Gamma_1) \cup \mathcal{Dom}\big(\bigcup_{j=2}^{n}\Gamma_j\big)$, from Property 3.2. Consequently we find that $\mathcal{Dom}(\Gamma_1) \cup \mathcal{Dom}\big(\bigcup_{j=2}^{n}\Gamma_j\big) = \mathcal{Dom}(\Gamma_1) \cup \big(\bigcup_{j=2}^{n}\mathcal{Dom}(\Gamma_j)\big)$, by induction hypothesis. Hence $\mathcal{Dom}\big(\bigcup_{j=1}^{n}\Gamma_j\big) = \mathcal{Dom}(\Gamma_1) \cup \big(\bigcup_{j=2}^{n}\mathcal{Dom}(\Gamma_j)\big) = \bigcup_{j=1}^{n}\mathcal{Dom}(\Gamma_j)$.

$\square$

Definition 3.8 (matching rules on pure canonical terms) *We define the matching rules on canonical terms as follow, where canonical terms t range over $\mathbb{T}$, and patterns $\mathcal{R}$ over $\mathbb{R}$.*

$$\frac{\overline{(t_0, \mathcal{R}_0) \in \Gamma_{gnd}}}{\big\{(t_0, \mathcal{R}_0)\big\} \vdash t_0 : \mathcal{R}_0} \textit{ (T-Atom0-canon)} \qquad \frac{\overline{(t_1, \mathcal{R}_1) \in \Gamma_{def}}}{\big\{(t_1, \mathcal{R}_1)\big\} \vdash t_1 : \mathcal{R}_1} \textit{ (T-Atom1-canon)}$$

$$\frac{\begin{array}{c}\Gamma_1 \vdash t_1 : \mathcal{R}_1 \quad \Gamma_2 \vdash t_2 : \mathcal{R}_2 \quad \mathcal{D}om(\Gamma_1) \cap \mathcal{D}om(\Gamma_2) = \phi \\ t_1 \wedge t_2 \notin \big(\mathcal{D}om(\Gamma_1) \cup \mathcal{D}om(\Gamma_2)\big)\end{array}}{\big\{(t_1 \wedge t_2, \mathcal{R}_1 \wedge \mathcal{R}_2)\big\} \cup (\Gamma_1 \cup \Gamma_2) \vdash t_1 \wedge t_2 : \mathcal{R}_1 \wedge \mathcal{R}_2} \textit{ (T-Cas-canon)}$$

$$\frac{\begin{array}{c}\Gamma_1 \vdash t_1 : \mathcal{R}_1 \quad \Gamma_2 \vdash t_2 : \mathcal{R}_2 \quad \mathcal{D}om(\Gamma_1) \cap \mathcal{D}om(\Gamma_2) = \phi \\ t_1 \vee t_2 \notin \big(\mathcal{D}om(\Gamma_1) \cup \mathcal{D}om(\Gamma_2)\big)\end{array}}{\big\{(t_1 \vee t_2, \mathcal{R}_1 \vee \mathcal{R}_2)\big\} \cup (\Gamma_1 \cup \Gamma_2) \vdash t_1 \vee t_2 : \mathcal{R}_1 \vee \mathcal{R}_2} \textit{ (T-Par-canon)}$$

$$\left.\begin{array}{ll} \dfrac{}{\Big\{\big(\{\}, \mathcal{R}_1{}^*\big)\Big\} \vdash \{\} : \mathcal{R}_1{}^* \quad \mathcal{R}_1 \textit{ is Any}} & (\textit{nil}) \\[2em] \dfrac{\begin{array}{c}\Gamma_1 \vdash t_1 : \mathcal{R}_1 \cdots \Gamma_n \vdash t_n : \mathcal{R}_1, \textit{ where } n \geq 1 \\ \mathcal{D}om(\Gamma_j) \cap \mathcal{D}om(\Gamma_k) = \phi \textit{ s.t. } j \neq k \quad \{t_1 \cdots t_n\} \notin \bigcup_{j=1}^{n} \mathcal{D}om(\Gamma_j)\end{array}}{\Big\{\big(\{t_1 \cdots t_n\}, \mathcal{R}_1{}^*\big)\Big\} \cup \bigcup_{j=1}^{n} \Gamma_j \vdash \{t_1 \cdots t_n\} : \mathcal{R}_1{}^*} & (\infty) \end{array}\right\} \textit{ (T-Cat0-canon)}$$

$$\frac{\begin{array}{c}\Gamma_1 \vdash t_1 : \mathcal{R}_1 \cdots \Gamma_n \vdash t_n : \mathcal{R}_1, \textit{ where } n \geq 1 \\ \mathcal{D}om(\Gamma_j) \cap \mathcal{D}om(\Gamma_k) = \phi \textit{ s.t. } j \neq k \quad [t_1 \cdots t_n] \notin \bigcup_{j=1}^{n} \mathcal{D}om(\Gamma_j)\end{array}}{\Big\{\big([t_1 \cdots t_n], \mathcal{R}_1{}^+\big)\Big\} \cup \bigcup_{j=1}^{n} \Gamma_j \vdash [t_1 \cdots t_n] : \mathcal{R}_1{}^+} \textit{ (T-Cat1-canon)}$$

$$\frac{\begin{array}{c}\Gamma_1 \vdash t_1 : \mathcal{R}_1 \cdots \Gamma_n \vdash t_n : \mathcal{R}_1 \textit{ where } n \geq 1 \\ \mathcal{D}om(\Gamma_j) \cap \mathcal{D}om(\Gamma_k) = \phi \textit{ s.t. } j \neq k \quad \langle t_1 \cdots t_n \rangle \notin \bigcup_{j=1}^{n} \mathcal{D}om(\Gamma_j)\end{array}}{\Big\{\big(\langle t_1 \cdots t_n \rangle, \mathcal{R}_1{}^{\downarrow}\big)\Big\} \cup \bigcup_{j=1}^{n} \Gamma_j \vdash \langle t_1 \cdots t_n \rangle : \mathcal{R}_1{}^{\downarrow}} \textit{ (T-Dup-canon)}$$

$$\left.\begin{array}{ll}\dfrac{}{\big\{(\circ,\mathcal{R}_1{}^?)\big\}\vdash\circ:\mathcal{R}_1{}^?\quad\mathcal{R}_1\,\mathit{is\,Any}} & (\mathit{nil}) \\[2ex] \dfrac{\Gamma_1\vdash t_1:\mathcal{R}_1\quad t_1{}^?\notin\mathcal{D}om(\Gamma_1)}{\big\{(t_1{}^?,\mathcal{R}_1{}^?)\big\}\cup\Gamma_1\vdash t_1{}^?:\mathcal{R}_1{}^?} & (\mathit{sol})\end{array}\right\}\textit{(T-Opt-canon)}$$

$$\left.\begin{array}{ll}\dfrac{\Gamma_1\vdash t_1:\mathcal{R}_1\quad\mathcal{R}_2\,\mathit{is\,Any}\quad t_1{}^{\leftarrow}\notin\mathcal{D}om(\Gamma_1)}{\big\{(t_1{}^{\leftarrow},\mathcal{R}_1|\mathcal{R}_2)\big\}\cup\Gamma_1\vdash t_1{}^{\leftarrow}:\mathcal{R}_1|\mathcal{R}_2} & (L) \\[2ex] \dfrac{\mathcal{R}_1\,\mathit{is\,Any}\quad\Gamma_1\vdash t_1:\mathcal{R}_2\quad t_1{}^{\rightarrow}\notin\mathcal{D}om(\Gamma_1)}{\big\{(t_1{}^{\rightarrow},\mathcal{R}_1|\mathcal{R}_2)\big\}\cup\Gamma_1\vdash t_1{}^{\rightarrow}:\mathcal{R}_1|\mathcal{R}_2} & (R)\end{array}\right\}\textit{(T-Alt-canon)}$$

From def. of pattern matching rules, next 2 lemmas are straitforward.

Lemma 3.1 *termination on canonical pattern matching.*

$$\Gamma\vdash t:\mathcal{R}\textit{ has termination.}$$

Proof. *Proof is directly structual induction on $\mathcal{R}$ of $\Gamma\vdash t:\mathcal{R}$.* $\square$

Property 3.4

$$\mathcal{D}om(\Gamma)=\mathit{sub}(t)\textit{ where }\Gamma\vdash t:\mathcal{R}.$$

Proof. *Proof is simple induction on structure of t s.t. $\Gamma\vdash t:\mathcal{R}$. Here we show the proof only for the most significant case.*

Case (T-Cat0-canon-∞): From def. of (T-Cat0-canon-∞), we let $\Gamma=\Big\{\big(\{t_1\cdots t_n\},\mathcal{R}_1{}^*\big)\Big\}\cup\bigcup_{j=1}^n\Gamma_j$. And from the facts of $\Gamma_1\vdash t_1:\mathcal{R}_1\cdots\Gamma_n\vdash t_n:\mathcal{R}_1$ comming from def. of (T-Cat0-canon-∞), we can find that $\mathcal{D}om(\Gamma_1)=\mathit{sub}(t_1)\cdots\mathcal{D}om(\Gamma_n)=\mathit{sub}(t_n)$ by I.H. Accounting on the fact of $\mathcal{D}om(\Gamma_1)\cup\cdots\cup\mathcal{D}om(\Gamma_n)=\bigcup_{j=1}^n\mathcal{D}om(\Gamma_j)$, we can express it as $\bigcup_{j=1}^n\mathcal{D}om(\Gamma_j)=\mathit{sub}(t_1)\cup\cdots\cup\mathit{sub}(t_n)$. By now we can show that $\mathcal{D}om(\Gamma)=\mathcal{D}om\bigg(\Big\{\big(\{t_1\cdots t_n\},\mathcal{R}_1{}^*\big)\Big\}\cup\bigcup_{j=1}^n\Gamma_j\bigg)=\mathcal{D}om\bigg(\Big\{\big(\{t_1\cdots t_n\},\mathcal{R}_1{}^*\big)\Big\}\bigg)\cup\mathcal{D}om\big(\bigcup_{j=1}^n\Gamma_j\big)=\big\{\{t_1\cdots t_n\}\big\}\cup\bigcup_{j=1}^n\mathcal{D}om(\Gamma_j)=$

$\{\{t_1 \cdots t_n\}\} \cup (\mathit{sub}(t_1) \cup \cdots \cup \mathit{sub}(t_n)) = \mathit{sub}(\{t_1 \cdots t_n\})$ *as desired, from def. of* $\mathcal{Dom}(\Gamma)$ *and* $\mathit{sub}(\{t_1 \cdots t_n\})$. $\square$

Property 3.5

$$\mathit{size}(t) \geq \mathit{size}(t') \text{ s.t. } t' \in \mathcal{Dom}(\Gamma), \text{ where } \Gamma \vdash t : \mathcal{R}.$$

Proof. *Proof is directly from def. of size of terms, with Property 3.4.* $\square$

Based on the matching rules above, we show the matching of $t_{TLSR_{4T}} : \mathcal{R}_{TLSR}$ introduced by Figure 2 as Figure 3.

$$
\dfrac{
(Par)\,\dfrac{
(0)\,\dfrac{\dfrac{}{(TLSR_{2T}, TLSR) \in \Gamma_{gnd}}}{\begin{array}{c}\Gamma_{10} \vdash TLSR_{2T} : TLSR\\ \Gamma_{10} = \{TLSR_{2T}, TLSR\}\end{array}}
\quad
(?)\,\dfrac{(L)\,\dfrac{(0)\,\dfrac{\dfrac{}{(NKR_{\#2}, NKR) \in \Gamma_{gnd}}}{\begin{array}{c}\Gamma_{13} \vdash NKR_{\#2} : NKR\\ \Gamma_{13} = \{NKR_{\#2}, NKR\}\end{array}}}{\begin{array}{c}\Gamma_{12} \vdash NKR_{\#2} : NKR|RKR\\ \Gamma_{12} = \{NKR_{\#2}, NKR|RKR\} \cup \Gamma_{13}\end{array}}}{\begin{array}{c}\Gamma_{11} \vdash NKR_{\#2}{}^{?} : (NKR|RKR)^{?}\\ \Gamma_{11} = \{NKR_{\#2}{}^{?}, (NKR|RKR)^{?}\} \cup \Gamma_{12}\end{array}}
}{\begin{array}{c}\Gamma_8 \vdash TLSR_{2T} \vee NKR_{\#2}{}^{?} : TLSR \vee (NKR|RKR)^{?}\\ \Gamma_8 = \{TLSR_{2T} \vee NKR_{\#2}{}^{?}, TLSR \vee (NKR|RKR)^{?}\} \cup (\Gamma_{10} \cup \Gamma_{11})\end{array}}
\quad
\dfrac{
\dfrac{\dfrac{}{(TLSR_{3T}, TLSR) \in \Gamma_{gnd}}}{\begin{array}{c}\Gamma_{14} \vdash TLSR_{3T} : TLSR\\ \Gamma_{14} = \{TLSR_{3T}, TLSR\}\end{array}}\,(0)
\quad
\dfrac{\dfrac{\dfrac{\dfrac{}{(RKR_{\#3}, RKR) \in \Gamma_{gnd}}}{\begin{array}{c}\Gamma_{17} \vdash RKR_{\#3}, RKR\\ \Gamma_{17} = \{RKR_{\#3}, RKR\}\end{array}}\,(0)}{\begin{array}{c}\Gamma_{16} \vdash RKR_{\#3} : NKR|RKR\\ \Gamma_{16} = \{RKR_{\#3}, NKR|RKR\} \cup \Gamma_{17}\end{array}}\,(R)}{\begin{array}{c}\Gamma_{15} \vdash RKR_{\#3}{}^{?} : (NKR|RKR)^{?}\\ \Gamma_{15} = \{RKR_{\#3}{}^{?}, (NKR|RKR)^{?}\} \cup \Gamma_{16}\end{array}}\,(?)
}{\begin{array}{c}\Gamma_9 \vdash TLSR_{3T} \vee RKR_{\#3}{}^{?} : TLSR \vee (NKR|RKR)^{?}\\ \Gamma_9 = \{TLSR_{3T} \vee RKR_{\#3}{}^{?}, TLSR \vee (NKR|RKR)^{?}\} \cup (\Gamma_{14} \cup \Gamma_{15})\end{array}}\,(Par)
}{\begin{array}{c}\Gamma_6 \vdash \{(TLSR_{2T} \vee NKR_{\#2}{}^{?}), (TLSR_{3T} \vee RKR_{\#3}{}^{?})\} : (TLSR \vee (NKR|RKR)^{?})^{*}\\ \Gamma_6 = \Big\{\{(TLSR_{2T} \vee NKR_{\#2}{}^{?}), (TLSR_{3T} \vee RKR_{\#3}{}^{?})\}, (TLSR \vee (NKR|RKR)^{?})^{*}\Big\} \cup (\Gamma_8 \cup \Gamma_9)\end{array}}\,(\infty)
$$

$$
\dfrac{
\Gamma_6 \vdash \{(TLSR_{2T} \vee NKR_{\#2}{}^{?}), (TLSR_{3T} \vee RKR_{\#3}{}^{?})\} : (TLSR \vee (NKR|RKR)^{?})^{*}
\quad
\dfrac{(0)\,\dfrac{\dfrac{}{(TLSR_{4T}, TLSR) \in \Gamma_{gnd}}}{\begin{array}{c}\Gamma_{18} \vdash TLSR_{4T} : TLSR\\ \Gamma_{18} = \{TLSR_{4T}, TLSR\}\end{array}}
\quad
(0)\,\dfrac{\dfrac{}{(TR_4, TR) \in \Gamma_{gnd}}}{\begin{array}{c}\Gamma_{19} \vdash TR_4 : TR\\ \Gamma_{19} = \{TR_4, TR\}\end{array}}}{\begin{array}{c}\Gamma_7 \vdash TLSR_{4T} \vee TR_4 : TLSR \vee TR\\ \Gamma_7 = \{TLSR_{4T} \vee TR_4, TLSR \vee TR\} \cup (\Gamma_{18} \cup \Gamma_{19})\end{array}}\,(Par)
}{\begin{array}{c}\Gamma_3 \vdash \{(TLSR_{2T} \vee NKR_{\#2}{}^{?}), (TLSR_{3T} \vee RKR_{\#3}{}^{?})\} \wedge (TLSR_{4T} \vee TR_4) : (TLSR \vee (NKR|RKR)^{?})^{*} \wedge (TLSR \vee TR)\\ \Gamma_3 = \Big\{\{(TLSR_{2T} \vee NKR_{\#2}{}^{?}), (TLSR_{3T} \vee RKR_{\#3}{}^{?})\} \wedge (TLSR_{4T} \vee TR_4), (TLSR \vee (NKR|RKR)^{?})^{*} \wedge (TLSR \vee TR)\Big\} \cup (\Gamma_6 \cup \Gamma_7)\end{array}}\,(Cas)
$$

$$
\dfrac{
\dfrac{(0)\,\dfrac{\dfrac{}{(TLSR_{1T}, TLSR) \in \Gamma_{gnd}}}{\begin{array}{c}\Gamma_4 \vdash TLSR_{1T} : TLSR\\ \Gamma_4 = \{TLSR_{1T}, TLSR\}\end{array}}
\quad
(nil^{?})\,\dfrac{}{\begin{array}{c}\Gamma_5 \vdash \circ : (NKR|RKR)^{?}\\ \Gamma_5 = \{\circ, (NKR|RKR)^{?}\}\end{array}}}{\begin{array}{c}\Gamma_2 \vdash TLSR_{1T} \vee \circ : TLSR \vee (NKR|RKR)^{?}\\ \Gamma_2 = \Big\{TLSR_{1T} \vee \circ, TLSR \vee (NKR|RKR)^{?}\Big\} \cup (\Gamma_4 \cup \Gamma_5)\end{array}}\,(Par)
\quad
\Gamma_3 \vdash \{(TLSR_{2T} \vee NKR_{\#2}{}^{?}), (TLSR_{3T} \vee RKR_{\#3}{}^{?})\} \wedge (TLSR_{4T} \vee TR_4) : (TLSR \vee (NKR|RKR)^{?})^{*} \wedge (TLSR \vee TR)
}{\begin{array}{c}\Gamma_1 \vdash (TLSR_{1T} \vee \circ) \wedge \Big(\{(TLSR_{2T} \vee NKR_{\#2}{}^{?}), (TLSR_{3T} \vee RKR_{\#3}{}^{?})\} \wedge (TLSR_{4T} \vee TR_4)\Big) : (TLSR \vee (NKR|RKR)^{?}) \wedge \Big((TLSR \vee (NKR|RKR)^{?})^{*} \wedge (TLSR \vee TR)\Big)\\ \Gamma_1 = \Bigg\{(TLSR_{1T} \vee \circ) \wedge \Big(\{(TLSR_{2T} \vee NKR_{\#2}{}^{?}), (TLSR_{3T} \vee RKR_{\#3}{}^{?})\} \wedge (TLSR_{4T} \vee TR_4)\Big), (TLSR \vee (NKR|RKR)^{?}) \wedge \Big((TLSR \vee (NKR|RKR)^{?})^{*} \wedge (TLSR \vee TR)\Big)\Bigg\} \cup (\Gamma_2 \cup \Gamma_3)\end{array}}\,(Cas)
$$

legend (0): *(T-Atom0-canon)*, (Cas): *(T-Cas-canon)*, (Par): *(T-Par-canon)*, (∞): *(T-Cat0-canon-∞)*, $(nil^{?})$: *(T-Opt-canon-nil)*, $(?)$: *(T-Opt-canon-sol)*, (L): *(T-Alt-canon-L)*, (R): *(T-Alt-canon-R)*.

Figure 3: pattern matching of interlocking logic example 1

4 More flexible system

We have developed the basic system in previous chapter, but as you see, it's hard to use. Rigid discipline forces us to behave strictly for expressions. In short, excessive strictness also decreases its usability. E.g. $1 + (2 + 3)$ and $(1 + 2) + 3$ aren't identical syntactically, but it's natural to regard them as identical, since associativity has little importance on same precedence operations. Similar to that, it's preferable to match both instances of $(\mathit{TLSR}_{1T} \vee \circ) \wedge \Big(\big\{(\mathit{TLSR}_{2T} \vee \mathit{NKR}_{\#2}{}^{?}), (\mathit{TLSR}_{3T} \vee \mathit{RKR}_{\#3}{}^{?})\big\} \wedge (\mathit{TLSR}_{4T} \vee \mathit{TR}_4)\Big)$ and $\Big((\mathit{TLSR}_{1T} \vee \circ) \wedge \big\{(\mathit{TLSR}_{2T} \vee \mathit{NKR}_{\#2}{}^{?}), (\mathit{TLSR}_{3T} \vee \mathit{RKR}_{\#3}{}^{?})\big\}\Big) \wedge (\mathit{TLSR}_{4T} \vee \mathit{TR}_4)$, but pattern of $\big(\mathit{TLSR} \vee (\mathit{NKR}|\mathit{RKR})^{?}\big) \wedge \Big(\big(\mathit{TLSR} \vee (\mathit{NKR}|\mathit{RKR})^{?}\big)^{*} \wedge (\mathit{TLSR} \vee \mathit{TR})\Big)$ matches with only former one since their associativity embedded in their own structures. Accordingly, we hope to describe instances more simply, and match them more flexible. For example it's preferable that we can express more flexibly them above as $\mathit{TLSR}_{1T} \wedge (\mathit{TLSR}_{2T} \vee \mathit{NKR}_{\#2}) \wedge (\mathit{TLSR}_{3T} \vee \mathit{RKR}_{\#3}) \wedge (\mathit{TLSR}_{4T} \vee \mathit{TR}_4)$, and matches with the pattern of $\big(\mathit{TLSR} \vee (\mathit{NKR}|\mathit{RKR})^{?}\big)^{*} \wedge (\mathit{TLSR} \vee \mathit{TR})$ by accounting on semantic meaming, but it's also impossible on current system.

In this chapter, we bring our system more flexibility, and make it more usable based on a well known type theoretical technology as polymorphism.

As first step, we define relations of intuitionistic equivalence over terms, which allow us to regard distinct terms as equivalent, as follow.

Definition 4.1 (intuitionistic equivalence over canonical terms) *We define the set of intuitionistic equivalence terms of t as follows. We write $t \simeq t'$ if t is intuitionistic equivalent to t'.*

$$\frac{\overline{t \equiv t'}}{t \simeq t'} \quad (\textit{Identity})$$

$$\frac{t \simeq t'}{t' \simeq t} \quad (\textit{Symmetry}) \qquad \frac{t_1 \simeq t_2 \quad t_2 \simeq t_3}{t_1 \simeq t_3} \quad (\textit{Transitivity})$$

$$\frac{t_1 \simeq t_1' \quad t_2 \simeq t_2'}{t_1 \wedge t_2 \simeq t_1' \wedge t_2'} \quad (\textit{Transparency-Cas}) \qquad \frac{t_1 \simeq t_1' \quad t_2 \simeq t_2'}{t_1 \vee t_2 \simeq t_1' \vee t_2'} \quad (\textit{Transparency-Par})$$

$$\frac{t \simeq t_1 \wedge (t_2 \wedge t_3)}{t \simeq (t_1 \wedge t_2) \wedge t_3} \quad (\textit{Associativity-Cas}_{\mathcal{L}}) \qquad \frac{t \simeq (t_1 \wedge t_2) \wedge t_3}{t \simeq t_1 \wedge (t_2 \wedge t_3)} \quad (\textit{Associativity-Cas}_{\mathcal{R}})$$

$$\frac{t \simeq (t_1 \vee t_2) \vee t_3}{t \simeq t_1 \vee (t_2 \vee t_3)} \quad (\textit{Associativity-Par}_{\mathcal{L}}) \qquad \frac{t \simeq t_1 \vee (t_2 \vee t_3)}{t \simeq (t_1 \vee t_2) \vee t_3} \quad (\textit{Associativity-Par}_{\mathcal{R}})$$

$$\frac{t \simeq t'}{t \simeq \{t'\}} \quad (\textit{Associativity-Cat0}_1) \qquad \frac{t \simeq t_1 \wedge \{t_2 \cdots t_n\}}{t \simeq \{t_1, t_2 \cdots t_n\}} \quad (\textit{AssociativityH-Cat0}_\infty)$$

$$\frac{t \simeq t'}{t \simeq [t']} \quad (\textit{Associativity-Cat1}_1) \qquad \frac{t \simeq t_1 \wedge [t_2 \cdots t_n]}{t \simeq [t_1, t_2 \cdots t_n]} \quad (\textit{AssociativityH-Cat1}_\infty)$$

$$\frac{t \simeq t'}{t \simeq \langle t' \rangle} \quad (\textit{Associativity-Dup}_1) \qquad \frac{t \simeq t_1 \vee \langle t_2 \cdots t_n \rangle}{t \simeq \langle t_1, t_2 \cdots t_n \rangle} \quad (\textit{AssociativityH-Dup}_\infty)$$

$$\frac{t \simeq \{\,\} \wedge t'}{t \simeq t'} \quad (\textit{RevealH-Cas}_{\{\}}) \qquad \frac{t \simeq t' \wedge \{\,\}}{t \simeq t'} \quad (\textit{RevealT-Cas}_{\{\}})$$

$$\frac{t \simeq \circ \wedge t'}{t \simeq t'} \quad (\textit{RevealH-Cas}_{\circ}) \qquad \frac{t \simeq t' \wedge \circ}{t \simeq t'} \quad (\textit{RevealT-Cas}_{\circ})$$

$$\frac{t \simeq \{\,\} \vee t'}{t \simeq t'} \quad (\textit{RevealH-Par}_{\{\}}) \qquad \frac{t \simeq t' \vee \{\,\}}{t \simeq t'} \quad (\textit{RevealT-Par}_{\{\}})$$

$$\frac{t \simeq \circ \vee t'}{t \simeq t'} \quad (\textit{RevealH-Par}_{\circ}) \qquad \frac{t \simeq t' \vee \circ}{t \simeq t'} \quad (\textit{RevealT-Par}_{\circ})$$

$$\frac{t \simeq t'}{t \simeq \{\,\} \wedge t'}\ (\mathit{PhonyH\text{-}Cas}_{\{\}}) \qquad \frac{t \simeq t'}{t \simeq t' \wedge \{\,\}}\ (\mathit{PhonyT\text{-}Cas}_{\{\}})$$

$$\frac{t \simeq t'}{t \simeq \circ \wedge t'}\ (\mathit{PhonyH\text{-}Cas}_{\circ}) \qquad \frac{t \simeq t'}{t \simeq t' \wedge \circ}\ (\mathit{PhonyT\text{-}Cas}_{\circ})$$

$$\frac{t \simeq t'}{t \simeq \{\,\} \vee t'}\ (\mathit{PhonyH\text{-}Par}_{\{\}}) \qquad \frac{t \simeq t'}{t \simeq t' \vee \{\,\}}\ (\mathit{PhonyT\text{-}Par}_{\{\}})$$

$$\frac{t \simeq t'}{t \simeq \circ \vee t'}\ (\mathit{PhonyH\text{-}Par}_{\circ}) \qquad \frac{t \simeq t'}{t \simeq t' \vee \circ}\ (\mathit{PhonyT\text{-}Par}_{\circ})$$

$$\frac{t \simeq t'}{t \simeq t'^{?}}\ (\mathit{Optional}) \qquad \frac{t \simeq t'}{t \simeq t'^{\leftarrow}}\ (\mathit{Alt\text{-}L}) \qquad \frac{t \simeq t'}{t \simeq t'^{\rightarrow}}\ (\mathit{Alt\text{-}R})$$

Then we extend the definition of judgement to use the set of equivalent terms given from above definition of $t \simeq t_{equ}$.

Definition 4.2 (binding pentagram) *We extend the definition of bindings Γ defined above, and we call the extended ones range over Δ as binding pentagram.*

$$\Delta \overset{\text{def}}{:=} \big\{(t, t_{equ}, \mathcal{R}, \mathcal{S}_{\mathit{fin}}, \Delta')\big\} \cup \Delta' \ \ \mathit{s.t.}\ \Delta \vdash t \rhd \mathcal{R}$$
$$\qquad\qquad \mathit{where}\ \ t \simeq t_{equ}$$
$$\qquad\qquad\qquad \mathcal{S}_{\mathit{fin}} \in \{\mathit{gnd}, \mathit{def}, \wedge, \vee, \mathit{nil}, \mathit{sol}, \infty, L, R\}$$

We still regard extended bindings Δ as the mapping of $\mathbb{T} \mapsto \Delta$ as follow.

$$\Delta(t) \overset{\text{def}}{:=} \begin{cases} \big\{(t', t'_{equ}, \mathcal{R}, \mathcal{S}_{\mathit{fin}}, \Delta')\big\} & \mathit{if}\ t = t' \\ \Delta'(t) \quad \mathit{if}\ t \neq t',\ \Delta' \neq \phi \\ \phi \quad \mathit{if}\ t \neq t',\ \Delta' = \phi \end{cases}$$

Definition 4.3 (binding domain extended) *We define a set of terms t and t_{equ} comming from Δ where $\mathcal{S}_0 \in \{\mathit{gnd}, \mathit{def}, \mathit{nil}\}$, as $\mathcal{Dom}(\Delta)$ s.t.*

$$\mathcal{Dom}(\Delta) \overset{\text{def}}{:=} \big\{t \mid (t, t_{equ}, \mathcal{R}, \mathcal{S}_{\mathit{fin}}, \Delta') \in \Delta\big\} \cup \big\{t_{equ} \mid (t, t_{equ}, \mathcal{R}, \mathcal{S}_0, \Delta') \in \Delta\big\}$$

And we extend the inference rules to accept intuitionistic equivalence as follows.

Definition 4.4 (matching rules on polymorphic equivalent terms) *We extend the matching rules on pure canonical terms, to enlarge the domain of pattern matching as follows.*

$$\frac{\overline{t \simeq t_0 \quad (t_0, \mathcal{R}_0) \in \Gamma_{gnd}}}{\big\{(t, t_0, \mathcal{R}_0, gnd, \phi)\big\} \vdash t \rhd \mathcal{R}_0} \quad \textit{(T-Atom0-xtend)}$$

$$\frac{\overline{t \simeq t_1 \quad (t_1, \mathcal{R}_1) \in \Gamma_{def}}}{\big\{(t, t_1, \mathcal{R}_1, def, \phi)\big\} \vdash t \rhd \mathcal{R}_1} \quad \textit{(T-Atom1-xtend)}$$

$$\frac{\begin{array}{c}\Delta_1 \vdash t_1 \rhd \mathcal{R}_1 \quad \Delta_2 \vdash t_2 \rhd \mathcal{R}_2 \quad \mathcal{D}om(\Delta_1) \cap \mathcal{D}om(\Delta_2) = \phi \\ t \simeq t_1 \wedge t_2 \textit{ s.t. } t,\ t_1 \wedge t_2 \notin \mathcal{D}om(\Delta_1 \cup \Delta_2)\end{array}}{\Big\{\big(t, t_1 \wedge t_2, \mathcal{R}_1 \wedge \mathcal{R}_2, \wedge, (\Delta_1 \cup \Delta_2)\big)\Big\} \cup (\Delta_1 \cup \Delta_2) \vdash t \rhd \mathcal{R}_1 \wedge \mathcal{R}_2} \quad \textit{(T-Cas-xtend)}$$

$$\frac{\begin{array}{c}\Delta_1 \vdash t_1 \rhd \mathcal{R}_1 \quad \Delta_2 \vdash t_2 \rhd \mathcal{R}_2 \quad \mathcal{D}om(\Delta_1) \cap \mathcal{D}om(\Delta_2) = \phi \\ t \simeq t_1 \vee t_2 \textit{ s.t. } t,\ t_1 \vee t_2 \notin \mathcal{D}om(\Delta_1 \cup \Delta_2)\end{array}}{\Big\{\big(t, t_1 \vee t_2, \mathcal{R}_1 \vee \mathcal{R}_2, \vee, (\Delta_1 \cup \Delta_2)\big)\Big\} \cup (\Delta_1 \cup \Delta_2) \vdash t \rhd \mathcal{R}_1 \vee \mathcal{R}_2} \quad \textit{(T-Par-xtend)}$$

$$\left.\begin{array}{rl}
\dfrac{\overline{t \simeq \{\} \quad \mathcal{R}_1 \textit{ is Any}}}{\Big\{\big(t, \{\}, \mathcal{R}_1^*, nil, \phi\big)\Big\} \vdash t \rhd {\mathcal{R}_1}^*} & (nil) \\[2ex]
\dfrac{\Delta' \vdash t' \rhd \mathcal{R}_1 \quad t \simeq t' \textit{ s.t. } t \notin \mathcal{D}om(\Delta')}{\big\{(t, t', {\mathcal{R}_1}^*, sol, \Delta')\big\} \cup \Delta' \vdash t \rhd {\mathcal{R}_1}^*} & (sol) \\[2ex]
\dfrac{\begin{array}{c}\Delta_h \vdash t_h \rhd \mathcal{R}_1 \quad \Delta_t \vdash t_t \rhd {\mathcal{R}_1}^* \quad \mathcal{D}om(\Delta_h) \cap \mathcal{D}om(\Delta_t) = \phi \\ t \simeq t_h \wedge t_t \\ \begin{cases} \{t,\ t_h \wedge t'_t\} \cap \mathcal{D}om(\Delta_h \cup \Delta_t) = \phi & \textit{if } \mathcal{S}_{fin_t} \in \{nil,\ sol\} \\ \{t,\ t_h \wedge t'_t,\ t'_t\} \cap \mathcal{D}om(\Delta_h \cup \Delta_t) = \phi & \textit{otherwise}\end{cases}\end{array}}{\begin{array}{c}\Big\{\big(t, t_h \wedge t'_t, {\mathcal{R}_1}^*, \infty, (\Delta_h \cup \Delta'_t)\big)\Big\} \cup (\Delta_h \cup \Delta'_t) \vdash t \rhd {\mathcal{R}_1}^* \\ \textit{where } \Delta_t(t_t) = (t_t, t'_t, {\mathcal{R}_1}^*, \mathcal{S}_{fin_t}, \Delta'_t) \textit{ s.t. } \mathcal{S}_{fin_t} \in \{sol, \infty\},\ t'_t \neq \{\}\end{array}} & (\infty)
\end{array}\right\} \quad \textit{(T-Cat0-xtend)}$$

$$\left.\begin{array}{c}
\dfrac{\Delta' \vdash t' \triangleright \mathcal{R}_1 \quad t \simeq t' \textit{ s.t. } t \notin \mathcal{D}om(\Delta')}{\left\{(t, t', {\mathcal{R}_1}^{+}, sol, \Delta')\right\} \cup \Delta' \vdash t \triangleright {\mathcal{R}_1}^{+}} \quad (sol) \\[3ex]
\dfrac{\begin{array}{c}\Delta_h \vdash t_h \triangleright \mathcal{R}_1 \quad \Delta_t \vdash t_t \triangleright {\mathcal{R}_1}^{+} \quad \mathcal{D}om(\Delta_h) \cap \mathcal{D}om(\Delta_t) = \phi \\ t \simeq t_h \wedge t_t \\ \begin{cases} \{t,\ t_h \wedge t'_t\} \cap \mathcal{D}om(\Delta_h \cup \Delta_t) = \phi & \textit{if } \mathcal{S}_{fin_t} = sol \\ \{t,\ t_h \wedge t'_t,\ t'_t\} \cap \mathcal{D}om(\Delta_h \cup \Delta_t) = \phi & \textit{otherwise} \end{cases}\end{array}}{\begin{array}{c}\left\{\left(t, t_h \wedge t'_t, {\mathcal{R}_1}^{+}, \infty, (\Delta_h \cup \Delta'_t)\right)\right\} \cup (\Delta_h \cup \Delta'_t) \vdash t \triangleright {\mathcal{R}_1}^{+} \\ \textit{where } \Delta_t(t_t) = (t_t, t'_t, {\mathcal{R}_1}^{+}, \mathcal{S}_{fin_t}, \Delta'_t) \textit{ s.t. } \mathcal{S}_{fin_t} \in \{sol, \infty\}\end{array}} \quad (\infty)
\end{array}\right\} \textit{(T-Cat1-xtend)}$$

$$\left.\begin{array}{c}
\dfrac{\Delta' \vdash t' \triangleright \mathcal{R}_1 \quad t \simeq t' \textit{ s.t. } t \notin \mathcal{D}om(\Delta')}{\left\{(t, t', {\mathcal{R}_1}^{\downarrow}, sol, \Delta')\right\} \cup \Delta' \vdash t \triangleright {\mathcal{R}_1}^{\downarrow}} \quad (sol) \\[3ex]
\dfrac{\begin{array}{c}\Delta_h \vdash t_h \triangleright \mathcal{R}_1 \quad \Delta_t \vdash t_t \triangleright {\mathcal{R}_1}^{\downarrow} \quad \mathcal{D}om(\Delta_h) \cap \mathcal{D}om(\Delta_t) = \phi \\ t \simeq t_h \vee t_t \\ \begin{cases} \{t,\ t_h \vee t'_t\} \cap \mathcal{D}om(\Delta_h \cup \Delta_t) = \phi & \textit{if } \mathcal{S}_{fin_t} = sol \\ \{t,\ t_h \vee t'_t,\ t'_t\} \cap \mathcal{D}om(\Delta_h \cup \Delta_t) = \phi & \textit{otherwise} \end{cases}\end{array}}{\begin{array}{c}\left\{\left(t, t_h \vee t'_t, {\mathcal{R}_1}^{\downarrow}, \infty, (\Delta_h \cup \Delta'_t)\right)\right\} \cup (\Delta_h \cup \Delta'_t) \vdash t \triangleright {\mathcal{R}_1}^{\downarrow} \\ \textit{where } \Delta_t(t_t) = (t_t, t'_t, {\mathcal{R}_1}^{\downarrow}, \mathcal{S}_{fin_t}, \Delta'_t) \textit{ s.t. } \mathcal{S}_{fin_t} \in \{sol, \infty\}\end{array}} \quad (\infty)
\end{array}\right\} \textit{(T-Dup-xtend)}$$

$$\left.\begin{array}{c}
\dfrac{\overline{t \simeq \circ \quad \mathcal{R}_1 \textit{ is Any}}}{\left\{(t, \circ, {\mathcal{R}_1}^{?}, nil, \phi)\right\} \vdash t \triangleright {\mathcal{R}_1}^{?}} \quad (nil) \\[3ex]
\dfrac{\Delta' \vdash t' \triangleright \mathcal{R}_1 \quad t \simeq t' \textit{ s.t. } t \notin \mathcal{D}om(\Delta')}{\left\{(t, t', {\mathcal{R}_1}^{?}, sol, \Delta')\right\} \cup \Delta' \vdash t \triangleright {\mathcal{R}_1}^{?}} \quad (sol)
\end{array}\right\} \textit{(T-Opt-xtend)}$$

$$\left.\begin{array}{c}
\dfrac{\Delta' \vdash t' \triangleright \mathcal{R}_1 \quad \mathcal{R}_2 \textit{ is Any.} \quad t \simeq t' \textit{ s.t. } t \notin \mathcal{D}om(\Delta')}{\left\{(t, t', \mathcal{R}_1 | \mathcal{R}_2, L, \Delta')\right\} \cup \Delta' \vdash t \triangleright \mathcal{R}_1 | \mathcal{R}_2} \quad (L) \\[3ex]
\dfrac{\mathcal{R}_1 \textit{ is Any.} \quad \Delta'' \vdash t' \triangleright \mathcal{R}_2 \quad t \simeq t' \textit{ s.t. } t \notin \mathcal{D}om(\Delta'')}{\left\{(t, t', \mathcal{R}_1 | \mathcal{R}_2, R, \Delta'')\right\} \cup \Delta'' \vdash t \triangleright \mathcal{R}_1 | \mathcal{R}_2} \quad (R)
\end{array}\right\} \textit{(T-Alt-xtend)}$$

Lemma 4.1 *soundness on extended matching*

$$\Delta \vdash t \triangleright \mathcal{R} \textit{ with } \mathcal{D}om(\Delta) = \mathcal{D}om(\Gamma) \textit{ for some } \Delta, \textit{ s.t. } \Gamma \vdash t : \mathcal{R}$$

Proof. *Proof is induction on structure of term t s.t.* $\Gamma \vdash t : \mathcal{R}$ *with Claim 4.1.*

Case (T-Atom0-canon): Proof for this case is trivial, which is directly from both def. of (T-Atom0-canon) and (T-Atom0-xtend) as follow. Here we can assume that t *satisfies* $(t, \mathcal{R}_0) \in \Gamma_{gnd}$ *from def. of (T-Atom0-canon) , then we can derive the equation of* $\big\{(t, t, \mathcal{R}_0, gnd, \phi)\big\} \vdash t \rhd \mathcal{R}_1$ *from (T-Atom0-xtend) for* $t \simeq t$ *s.t.* $(t, \mathcal{R}_0) \in \Gamma_{gnd}$ *from (Identity), which satisfies the condition of* $\mathcal{D}om\Big(\big\{(t, \mathcal{R}_0)\big\}\Big) = \{t\} = \big(\{t\} \cup \phi\big) = \mathcal{D}om\Big(\big\{(t, t, \mathcal{R}_0, gnd, \phi)\big\}\Big)$ *as desired.*

Case (T-Atom1-canon): Proof is same scheme as the case of (T-Atom0-canon).

Case (T-Cas-canon): From induction hypothesis, we suppose that $\Delta_1 \vdash t_1 \rhd \mathcal{R}_1$, $\Delta_2 \vdash t_2 \rhd \mathcal{R}_2$ *with* $\mathcal{D}om(\Delta_1) = \mathcal{D}om(\Gamma_1)$ *and* $\mathcal{D}om(\Delta_2) = \mathcal{D}om(\Gamma_2)$ *respectively, which implies the fact of* $\mathcal{D}om(\Delta_1) \cap \mathcal{D}om(\Delta_2) = \phi$ *for* $\mathcal{D}om(\Gamma_1) \cap \mathcal{D}om(\Gamma_2) = \phi$. *Similarly we still find the fact that* $t_1 \wedge t_2$ *isn't belonging to* $\big(\mathcal{D}om(\Gamma_1) \cup \mathcal{D}om(\Gamma_1)\big) = \big(\mathcal{D}om(\Delta_1) \cup \mathcal{D}om(\Delta_1)\big)$. *Therefore we obtain the deduction of* $\Big\{\big(t_1 \wedge t_2, t_1 \wedge t_2, \mathcal{R}_1 \wedge \mathcal{R}_2, (\Delta_1 \cup \Delta_2)\big)\Big\} \cup (\Delta_1 \cup \Delta_2) \vdash t_1 \wedge t_2 \rhd \mathcal{R}_1 \wedge \mathcal{R}_2$ *from (T-Cas-xtend), with the condition of* $\mathcal{D}om\Bigg(\Big\{\big(t_1 \wedge t_2, t_1 \wedge t_2, \mathcal{R}_1 \wedge \mathcal{R}_2, (\Delta_1 \cup \Delta_2)\big)\Big\} \cup (\Delta_1 \cup \Delta_2)\Bigg) = \big(\{t_1 \wedge t_2\} \cup \mathcal{D}om(\Delta_1 \cup \Delta_2)\big) = \Big(\{t_1 \wedge t_2\} \cup \big(\mathcal{D}om(\Delta_1) \cup \mathcal{D}om(\Delta_2)\big)\Big) = \Big(\{t_1 \wedge t_2\} \cup \big(\mathcal{D}om(\Gamma_1) \cup \mathcal{D}om(\Gamma_2)\big)\Big) = \big(\{t_1 \wedge t_2\} \cup \mathcal{D}om(\Gamma_1 \cup \Gamma_2)\big) = \mathcal{D}om(\Gamma)$ *as desired.*

Case (T-Par-canon): Proof is same scheme as the case of (T-Cat-canon),

Case (T-Cat0-canon-nil), (T-Cat0-canon-∞*): Proof for these cases are from Claim 4.1.*

Case (T-Cat1-canon): Proof is same scheme to the case of (T-Cat0-canon-∞*).*

Case (T-Dup-canon): Proof is same scheme to the case of (T-Cat1-canon).

Case (T-Opt-canon): Proof is similar to the case of (T-Cat-canon-nil).

Case (T-Alt-canon-L): In this case, we suppose $\big\{(t_1{}^{\leftarrow}, \mathcal{R}_1|\mathcal{R}_1)\big\} \cup \Gamma_1 \vdash t_1{}^{\leftarrow} : \mathcal{R}_1|\mathcal{R}_1$, *then by induction hypothesis we also find that* $\Delta_1 \vdash t_1 \rhd \mathcal{R}_1$ *satisfying the condition of* $\mathcal{D}om(\Delta_1) = \mathcal{D}om(\Gamma_1)$ *s.t.* $t_1{}^{\leftarrow} \notin \mathcal{D}om(\Delta_1)$ *because of* $t_1{}^{\leftarrow} \notin \mathcal{D}om(\Gamma_1)$ *from def. of (T-Alt-canon-L). Hence we can derive* $\big\{(t_1{}^{\leftarrow}, t_1{}^{\leftarrow}, \mathcal{R}_1|\mathcal{R}_1, L, \Delta_1)\big\} \cup \Delta_1 \vdash t_1{}^{\leftarrow} \rhd \mathcal{R}_1|\mathcal{R}_1$ *from (T-Alt-xtend-L) with desired Property of* $\mathcal{D}om\Big(\big\{(t_1{}^{\leftarrow}, t_1{}^{\leftarrow}, \mathcal{R}_1|\mathcal{R}_1, L, \Delta_1)\big\} \cup \Delta_1\Big) = \{t_1{}^{\leftarrow}\} \cup \mathcal{D}om(\Delta_1) = \{t_1{}^{\leftarrow}\} \cup$

$\mathcal{D}om(\Gamma_1) = \mathcal{D}om(\Gamma)$.

Case (T-Alt-canon-R): Proof is similar to the case of (T-Alt-canon-L). $\square$

Claim 4.1

$$\Delta \vdash \{t_1, \cdots t_n\} \triangleright \mathcal{R} \textit{ with } \mathcal{D}om(\Delta) = \mathcal{D}om(\Gamma) \textit{ for some } \Delta, \textit{ s.t. } \Gamma \vdash \{t_1, \cdots t_n\} : \mathcal{R}$$

$$\textit{where the last applied rule on } \Delta \vdash \{t_1, \cdots t_n\} \triangleright \mathcal{R} \textit{ is } \begin{cases} \textit{(T-Cat0-xtend-nil)} & \textit{if } n = 0 \\ \textit{(T-Cat0-xtend-sol)} & \textit{if } n = 1 \\ \textit{(T-Cat0-xtend-}\infty\textit{)} & \textit{if } n \geq 2 \end{cases}$$

Proof. *Proof is induction on structure of t s.t.* $\Gamma \vdash t : \mathcal{R}$*, with Lemma 4.1.*

Case (T-Cat0-canon-nil): It's trivial, proof for this case is directly from def, of (T-Cat0-canon-nil) and (T-Cat0-xtend-nil),

Case (T-Cat0-canon-∞*): From induction hypothesis we can assume that* $\Delta_j \vdash t_j \triangleright \mathcal{R}_j$ *s.t.* $\mathcal{D}om(\Delta_j) = \mathcal{D}om(\Gamma_j)$ *for each* $1 \leq j \leq n$*. Now we examines for each cases of* $n = 1$ *and* $n \geq 2$*.* $(n = 1)$*: For first case, we can derive* $\{t_1\} \simeq t_1$ *by Claim 4.2, where we mention that* $\{t_1\} \notin \mathcal{D}om(\Delta_1) = \mathcal{D}om(\Gamma_1)$ *s.t.* $\Gamma_1 \vdash t_1 : \mathcal{R}_1$*. Thus we derive* $\Big\{\big(\{t_1\}, t_1, {\mathcal{R}_1}^*, sol, \Delta_1\big)\Big\} \cup \Delta_1 \vdash \{t_1\} \triangleright {\mathcal{R}_1}^*$ *from (T-Cat0-xtend-sol), with desired properties of* $\mathcal{D}om\bigg(\Big\{\big(\{t_1\}, t_1, {\mathcal{R}_1}^*, sol, \Delta_1\big)\Big\} \cup \Delta_1\bigg) = \big(\{t_1\} \cup \mathcal{D}om(\Delta_1)\big) = \big(\{t_1\} \cup \mathcal{D}om(\Gamma_1)\big) = \mathcal{D}om(\Gamma)$*, where we let* $\Gamma = \Big\{\big(\{t_1\}, {\mathcal{R}_1}^*\big)\Big\} \cup \Gamma_1$*.* $(n \geq 2)$*: Here we think of the case of* $n = 2$ *that* $\Big\{\big(\{t_1, t_2\}, {\mathcal{R}_1}^*\big)\Big\} \cup (\Gamma_1 \cup \Gamma_2) \vdash \{t_1, t_2\} : {\mathcal{R}_1}^*$*, where we are given several facts from induction hypotheses as* $\Delta_1 \vdash t_1 \triangleright \mathcal{R}_1$*,* $\Delta_2 \vdash t_2 \triangleright \mathcal{R}_2$*, s.t.* $\mathcal{D}om(\Delta_1) = \mathcal{D}om(\Gamma_1)$ *and* $\mathcal{D}om(\Delta_2) = \mathcal{D}om(\Gamma_2)$ *satisfying* $\mathcal{D}om(\Delta_1) \cap \mathcal{D}om(\Delta_2) = \phi$ *for* $\mathcal{D}om(\Gamma_1) \cap \mathcal{D}om(\Gamma_2) = \phi$ *respectively. And we can derive the term* $\{t_2\}$ *equivalent to* t_2 *i.e.* $\{t_2\} \simeq t_2$ *similar by Claim 4.2, with the condition of* $\{t_2\} \notin \mathcal{D}om(\Delta_2) = \mathcal{D}om(\Gamma_2)$ *s.t.* $\Gamma_2 \vdash t_2 : \mathcal{R}_1$ *for* $size\big(\{t_2\}\big) > size(t_2)$ *with Property 3.5. Then we derive* $\Delta_t \vdash \{t_2\} \triangleright {\mathcal{R}_1}^*$ *from (T-Cat0-xtend-sol), where we let* $\Delta_t = \Big\{\big(\{t_2\}, t_2, {\mathcal{R}_1}^*, sol, \Delta_2\big)\Big\} \cup \Delta_2$ *which sustains the condition of* $\mathcal{D}om(\Delta_1) \cap \mathcal{D}om(\Delta_t) = \phi$ *for both facts of* $\mathcal{D}om(\Delta_1) \cap \mathcal{D}om(\Delta_2) = \phi$ *and* $\{t_2\} \notin \mathcal{D}om(\Gamma_1) = \mathcal{D}om(\Delta_1)$*. Note that we assume* $\{t_2\} \in \mathcal{D}om(\Gamma_1)$ *whose binding* Γ_1 *comming from* $\Gamma_1 \vdash t_1 : \mathcal{R}_1$ *also satisfies the condition of* $\mathcal{D}om(\Gamma_1) = sub(t_1)$ *from Property 3.4, i.e.* $\{t_2\} \in sub(t_1)$ *implies* *that* $t_2 \in \mathcal{Dom}(\Gamma_1)$ *for* $t_2 \in \mathit{sub}\big(\{t_2\}\big) \subseteq \mathit{sub}(t_1) = \mathcal{Dom}(\Gamma_1)$ *from Property 3.1, which contradicts* $\mathcal{Dom}(\Gamma_1) \cap \mathcal{Dom}(\Gamma_2) = \phi$*. Consequently, we can derive the equivalency* $\{t_1, t_2\} \simeq t_1 \wedge \{t_2\}$*, which derivation's detail is explained in Claim 4.2, where both* $\{t_1, t_2\}$ *and* $t_1 \wedge t_2$ *aren't belonging to* $\mathcal{Dom}(\Delta_1 \cup \Delta_t) = \big(\mathcal{Dom}(\Delta_1) \cup \mathcal{Dom}(\Delta_t)\big) = \Big(\mathcal{Dom}(\Delta_1) \cup \big(\{t_2\} \cup \mathcal{Dom}(\Delta_2)\big)\Big) = \Big(\big(\mathcal{Dom}(\Gamma_1) \cup \mathcal{Dom}(\Gamma_2)\big) \cup \{t_2\}\Big)$ *for their size constraints. Then we eventually derive our goal of this case as* $\Big\{\big(\{t_1, t_2\}, t_1 \wedge t_2, \mathcal{R}_1{}^*, \infty, (\Delta_1 \cup \Delta_2)\big)\Big\} \cup (\Delta_1 \cup \Delta_2) \vdash \{t_1, t_2\} \triangleright \mathcal{R}_1{}^*$ *from (T-Cat0-xtend-∞). That also satisfies the condition of* $\mathcal{Dom}\Bigg(\Big\{\big(\{t_1, t_2\}, t_1 \wedge t_2, \mathcal{R}_1{}^*, \infty, (\Delta_1 \cup \Delta_2)\big)\Big\} \cup (\Delta_1 \cup \Delta_2)\Bigg) = \big(\{t_1, t_2\} \cup \mathcal{Dom}(\Delta_1 \cup \Delta_2)\big) = \big(\{t_1, t_2\} \cup \mathcal{Dom}(\Gamma_1 \cup \Gamma_2)\big) = \mathcal{Dom}(\Gamma)$ *as desired, where we let* $\Gamma = \Big\{\big(\{t_1, t_2\}, \mathcal{R}_1{}^*\big)\Big\} \cup (\Gamma_1 \cup \Gamma_2)$*. Next we think of the case of* $n > 2$*. In this case we suppose the situation of* $\Big\{\big(\{t_1 \cdots t_n\}, \mathcal{R}_1{}^*\big)\Big\} \cup \bigcup_{j=1}^{n} \Gamma_j \vdash \{t_1 \cdots t_n\} : \mathcal{R}_1{}^*$*, which implies the facts that* $\Gamma_2 \vdash t_2 : \mathcal{R}_1 \cdots \Gamma_n \vdash t_n : \mathcal{R}_1$ *satisfying the conditions of* $\mathcal{Dom}(\Gamma_j) \cap \mathcal{Dom}(\Gamma_k) = \phi$ *if* $j \neq k$*. Accounting on the fact of* $\{t_2 \cdots t_n\} \notin \bigcup_{j=2}^{n} \mathcal{Dom}(\Gamma_j)$ *since term size of* $\{t_2 \cdots t_n\}$ *is greater than any of* $t_2 \cdots t_n$ *s.t.* $\Gamma_2 \vdash t_2 : \mathcal{R}_1 \cdots \Gamma_n \vdash t_n : \mathcal{R}_1$ *respectively, we can derive* $\Big\{\big(\{t_2 \cdots t_n\}, \mathcal{R}_1{}^*\big)\Big\} \cup \bigcup_{j=2}^{n} \Gamma_j \vdash \{t_2 \cdots t_n\} : \mathcal{R}_1{}^*$ *from (T-Cat0-canon). Here we let n be the number of subterms of* $\{t_1 \cdots t_n\}$*, the cases of* $n \in \{1, 2\}$ *are shown as the cases of* $n \leq 2$ *above. For the case of* $n > 2$*, we suppose the derivation* $\Big\{\big(\{t_2 \cdots t_n\}, \mathcal{R}_1{}^*\big)\Big\} \cup \bigcup_{j=2}^{n} \Gamma_j \vdash \{t_2 \cdots t_n\} : \mathcal{R}_1{}^*$*, which gives us the fact of* $\Delta_t \vdash \{t_2 \cdots t_n\} \triangleright \mathcal{R}_1{}^*$ *for some* Δ_t *satisfying the condition of* $\mathcal{Dom}(\Delta_t) = \mathcal{Dom}(\Gamma_t)$*, where we let* $\Gamma_t = \Big\{\big(\{t_2 \cdots t_n\}, \mathcal{R}_1{}^*\big)\Big\} \cup \bigcup_{j=2}^{n} \Gamma_j$ *by induction hypothesis. Now we mention again that the prerequisite of* $\Big\{\big(\{t_1 \cdots t_n\}, \mathcal{R}_1{}^*\big)\Big\} \cup \bigcup_{j=1}^{n} \Gamma_j \vdash \{t_1 \cdots t_n\} : \mathcal{R}_1{}^*$ *includes the fact of* $\Gamma_1 \vdash t_1 : \mathcal{R}_1$*, which also brings us* $\Delta_1 \vdash t_1 \triangleright \mathcal{R}_1$ *with the condition of* $\mathcal{Dom}(\Delta_1) = \mathcal{Dom}(\Gamma_1)$ *by induction hypothesis. Thus we can still find the Property of* $\mathcal{Dom}(\Delta_1) \cap \mathcal{Dom}(\Delta_t) = \mathcal{Dom}(\Gamma_1) \cap \big(\{t_2 \cdots t_n\} \cup \bigcup_{j=2}^{n} \mathcal{Dom}(\Gamma_j)\big) = \phi$ *since we assume* $\{t_2 \cdots t_n\} \in \mathcal{Dom}(\Gamma_1)$*, it seems to be* $t_j \in \mathit{sub}\big(\{t_2 \cdots t_n\}\big) \subseteq \mathcal{Dom}(\Gamma_1)$ *s.t.* $2 \leq j \leq n$*, which condtradicts* $\mathcal{Dom}(\Gamma_1) \cap \mathcal{Dom}(\Gamma_j) = \phi$ *where* $2 \leq j \leq n$*. From similar reason both* $\{t_1, \cdots t_n\}$

and $t_1 \wedge t'_t$ aren't belonging to $\mathcal{Dom}(\Gamma_1) \cup \big(\{t_2 \cdots t_n\} \cup \bigcup_{j=2}^{n} \mathcal{Dom}(\Gamma_j)\big) = \mathcal{Dom}(\Delta_1 \cup \Delta_t)$, since $t_1 \wedge t'_t \notin \mathcal{Dom}(\Gamma_1)$ is by term size constraint, and each $t_1 \notin \mathcal{Dom}(\Gamma_j)$ implises $t_1 \wedge t'_t \notin \big(\{t_2 \cdots t_n\} \cup \bigcup_{j=2}^{n} \mathcal{Dom}(\Gamma_j)\big)$, i.e. assumimg of $t_1 \wedge t'_t \in \mathcal{Dom}(\Gamma_j)$ for some j imposes on $t_1 \in \mathcal{Dom}(\Gamma_j) = \mathcal{sub}(t_j)$ s.t. $\Gamma_j \vdash t_j : \mathcal{R}_1$ because of $t_1 \in \mathcal{sub}(t_1 \wedge t'_t)$ from Property 3.1, it contradicts the fact of $\mathcal{Dom}(\Gamma_1) \cap \mathcal{Dom}(\Gamma_j) = \phi$. Accounting on the fact that the last applied rule on $\Delta_t \vdash \{t_2 \cdots t_n\} \triangleright {\mathcal{R}_1}^*$, $n > 2$ is (T-Cat0-xtend-∞) by induction hypothesis, where we find $t'_t \notin \mathcal{Dom}(\Delta_t)$ because of $t_t = \{t_2 \cdots t_n\} \neq t'_t$. We even can let $t'_t = t_{t_h} \wedge t'_{t_t}$ s.t. $\Delta_{t_h} \vdash t_{t_h} \triangleright \mathcal{R}_1$, thus $t'_t \notin \mathcal{Dom}(\Delta_1)$. We assume $t'_t \in \mathcal{Dom}(\Delta_1) = \mathcal{Dom}(\Gamma_1) = \mathcal{sub}(t_1)$, then we find $t_{t_h} \in \mathcal{sub}(t'_t) \subseteq \mathcal{sub}(t_1) = \mathcal{Dom}(\Gamma_1)$, which contradicts $\mathcal{Dom}(\Gamma_1) \cap \bigcup_{j=2}^{n} \mathcal{Dom}(\Gamma_j) = \phi$. Since $t_{t_h} \in \mathcal{Dom}(\Delta_{t_h})$ s.t. $\Delta_{t_h} \vdash t_{t_h} \triangleright \mathcal{R}_1$ implies $t_{t_h} \neq \{t_2, \cdots t_n\}$ for $\{t_2, \cdots t_n\} \notin \mathcal{Dom}(\Delta_{t_h} \cup \Delta'_{t_t})$, while $t_{t_h} \in \mathcal{Dom}(\Delta_{t_h} \cup \Delta'_{t_t}) \subseteq \mathcal{Dom}(\Delta_t)$ s.t. $\mathcal{Dom}(\Delta_t) = \mathcal{Dom}(\Gamma_t) = \Big(\big\{\{t_2 \cdots t_n\}\big\} \cup \bigcup_{j=2}^{n} \mathcal{Dom}(\Gamma_j)\Big)$ from def. of (T-Cat0-xtend-∞). Therefore $\{t_{t_h}\} \cap \mathcal{Dom}(\Gamma_t) = \{t_{t_h}\} \cap \Big(\big\{\{t_2 \cdots t_n\}\big\} \cap \bigcup_{j=2}^{n} \mathcal{Dom}(\Gamma_j)\Big) = \{t_{t_h}\} \cap \bigcup_{j=2}^{n} \mathcal{Dom}(\Gamma_j) \supseteq \{t_{t_h}\}$ s.t. $t_{t_h} \in \mathcal{Dom}(\Gamma_1)$ on our assumption, which contradicts the fact of $\mathcal{Dom}(\Gamma_1) \cap \mathcal{Dom}\big(\bigcup_{j=2}^{n} \Gamma_j\big) = \phi$. Consequently we obtain $t'_t \notin \mathcal{Dom}(\Delta_1 \cup \Delta_t)$. According to the facts we have revealed above, we can derive our conclusion as $\Big\{\big(\{t_1 \cdots t_n\}, t_1 \wedge t'_t, {\mathcal{R}_1}^*, \infty, (\Delta_1 \cup \Delta'_t)\big)\Big\} \cup (\Delta_1 \cup \Delta'_t) \vdash \{t_1 \cdots t_n\} \triangleright {\mathcal{R}_1}^*$ s.t. $\Delta'_t = \Delta_t \setminus \Delta_t\big(\{t_2 \cdots t_n\}\big)$ where $\Delta_t\big(\{t_2 \cdots t_n\}\big) = \big(\{t_2 \cdots t_n\}, t'_t, {\mathcal{R}_1}^*, \infty, \Delta'_t\big)$ from (T-Cat0-xtend-∞), with the Property of $\mathcal{Dom}\bigg(\Big\{\big(\{t_1 \cdots t_n\}, t_1 \wedge t'_t, {\mathcal{R}_1}^*, \infty, (\Delta_1 \cup \Delta'_t)\big)\Big\} \cup (\Delta_1 \cup \Delta'_t)\bigg) = \big\{\{t_1 \cdots t_n\}\big\} \cup \mathcal{Dom}(\Delta_1 \cup \Delta'_t) = \big\{\{t_1 \cdots t_n\}\big\} \cup \big(\mathcal{Dom}(\Delta_1) \cup \mathcal{Dom}(\Delta'_t)\big) = \big\{\{t_1 \cdots t_n\}\big\} \cup \bigg(\mathcal{Dom}(\Delta_1) \cup \mathcal{Dom}\Big(\Delta_t \setminus \Delta_t\big(\{t_2 \cdots t_n\}\big)\Big)\bigg) = \big\{\{t_1 \cdots t_n\}\big\} \cup \big(\mathcal{Dom}(\Gamma_1) \cup \bigcup_{j=2}^{n} \mathcal{Dom}(\Gamma_j)\big) = \big\{\{t_1 \cdots t_n\}\big\} \cup \bigcup_{j=1}^{n} \mathcal{Dom}(\Gamma_j) = \mathcal{Dom}(\Gamma)$ as desired. $\square$

Claim 4.2

$$\{t_1\} \simeq t_1, \quad \{t_1, t_2\} \simeq t_1 \wedge \{t_2\}$$

Proof. *See below derivation for each.*

$$(\textit{Symmetry})\,\dfrac{(\textit{Associativity-Cat0}_1)\,\dfrac{(\textit{Identity})\,\dfrac{\overline{t_1 \equiv t_1}}{t_1 \simeq t_1}}{t_1 \simeq \{t_1\}}}{\{t_1\} \simeq t_1} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\overline{t_1 \equiv t_1}}{t_1 \simeq t_1}\,(\textit{Identity}) \quad \dfrac{\overline{\{t_2\} \equiv \{t_2\}}}{\{t_2\} \simeq \{t_2\}}\,(\textit{Identity})}{t_1 \wedge \{t_2\} \simeq t_1 \wedge \{t_2\}}\,(\textit{Transparency-Cas})}{t_1 \wedge \{t_2\} \simeq \{t_1, t_2\}}\,(\textit{AssociativityH-Cat0}_\infty)}{\{t_1, t_2\} \simeq t_1 \wedge \{t_2\}}\,(\textit{Symmetry})$$

□

Definition 4.5 (term canonicalizer) *We define the term canonicalizer $\mathcal{M}$, which transduces the term matched on extended rules to corresponding canonical one, as follow. It also sustains equality over canonicalizing, see Property 4.1.*

$$\mathcal{M}\big(t, t_{equ}, \mathcal{R}, \mathcal{S}_{fin}, \Delta'\big) := \begin{cases}
t_{equ} \quad \textit{if } \mathcal{S}_{fin} \in \big\{gnd, def, nil\big\} \\
\mathcal{M}\big(\Delta'(t_1)\big) \wedge \mathcal{M}\big(\Delta'(t_2)\big) \quad \textit{if } \mathcal{S}_{fin} \in \{\wedge\} \textit{ and } t_{equ} = t_1 \wedge t_2. \\
\mathcal{M}\big(\Delta'(t_1)\big) \vee \mathcal{M}\big((\Delta'(t_2)\big) \quad \textit{if } \mathcal{S}_{fin} \in \{\vee\} \textit{ and } t_{equ} = t_1 \vee t_2. \\
\Big\{\mathcal{M}\big(\Delta'(t_{equ})\big)\Big\} \quad \textit{if } \mathcal{S}_{fin} \in \{sol\} \textit{ and } \mathcal{R} = \mathcal{R}_1{}^* \textit{ for some } \mathcal{R}_1 \\
\Big\{\mathcal{M}\big(\Delta'(t_{equ_h})\big), \mathcal{M}\big(\Delta'(t_{equ_t})\big)\Big\} \quad \textit{if } \mathcal{S}_{fin} \in \{\infty\},\ \mathcal{R} = \mathcal{R}_1{}^* \\
\quad \textit{s.t. } \Delta'(t_{equ_t}) = (t_{equ_t}, t'_{equ_t}, \mathcal{R}_1, \mathcal{S}'_{fin}, \Delta''),\ t_{equ} = t_{equ_h} \wedge t_{equ_t} \\
\Big\{\mathcal{M}\big(\Delta'(t_{equ_h})\big), t_2 \cdots t_n\Big\} \quad \textit{if } \mathcal{S}_{fin} \in \{\infty\},\ \mathcal{R} = \mathcal{R}_1{}^* \\
\quad \textit{where } \{t_2 \cdots t_n\} = \mathcal{M}_{\{\}}\Big(t_{equ_t}, \mathcal{R}, \Big(\Delta' \setminus \big(\Delta'(t_{equ_h}) \cup \Delta'_h\big)\Big)\Big) \\
\quad \textit{s.t. } t_{equ} = t_{equ_h} \wedge t_{equ_t} \textit{ and } \Delta'(t_{equ_h}) = (t_{equ_h}, t'_{equ_h}, \mathcal{R}_1, \mathcal{S}'_{fin}, \Delta'_h) \\
\Big[\mathcal{M}\big(\Delta'(t_{equ})\big)\Big] \quad \textit{if } \mathcal{S}_{fin} \in \{sol\} \textit{ and } \mathcal{R} = \mathcal{R}_1{}^+ \textit{ for some } \mathcal{R}_1 \\
\Big[\mathcal{M}\big(\Delta'(t_{equ_h})\big), \mathcal{M}\big(\Delta'(t_{equ_t})\big)\Big] \quad \textit{if } \mathcal{S}_{fin} \in \{\infty\},\ \mathcal{R} = \mathcal{R}_1{}^+ \\
\quad \textit{s.t. } \Delta'(t_{equ_t}) = (t_{equ_t}, t'_{equ_t}, \mathcal{R}_1, \mathcal{S}'_{fin}, \Delta''),\ t_{equ} = t_{equ_h} \wedge t_{equ_t} \\
\Big[\mathcal{M}\big(\Delta'(t_{equ_h})\big), t_2 \cdots t_n\Big] \quad \textit{if } \mathcal{S}_{fin} \in \{\infty\},\ \mathcal{R} = \mathcal{R}_1{}^+ \\
\quad \textit{where } [t_2 \cdots t_n] = \mathcal{M}_{[]}\Big(t_{equ_t}, \mathcal{R}, \Big(\Delta' \setminus \big(\Delta'(t_{equ_h}) \cup \Delta'_h\big)\Big)\Big) \\
\quad \textit{s.t. } t_{equ} = t_{equ_h} \wedge t_{equ_t} \textit{ and } \Delta'(t_{equ_h}) = (t_{equ_h}, t'_{equ_h}, \mathcal{R}_1, \mathcal{S}'_{fin}, \Delta'_h) \\
\Big\langle\mathcal{M}\big(\Delta'(t_{equ})\big)\Big\rangle \quad \textit{if } \mathcal{S}_{fin} \in \{sol\} \textit{ and } \mathcal{R} = \mathcal{R}_1{}^{\downarrow} \textit{ for some } \mathcal{R}_1 \\
\Big\langle\mathcal{M}\big(\Delta'(t_{equ_h})\big), \mathcal{M}\big(\Delta'(t_{equ_t})\big)\Big\rangle \quad \textit{if } \mathcal{S}_{fin} \in \{\infty\},\ \mathcal{R} = \mathcal{R}_1{}^{\downarrow} \\
\quad \textit{s.t. } \Delta'(t_{equ_t}) = (t_{equ_t}, t'_{equ_t}, \mathcal{R}_1, \mathcal{S}'_{fin}, \Delta''),\ t_{equ} = t_{equ_h} \vee t_{equ_t} \\
\Big\langle\mathcal{M}\big(\Delta'(t_{equ_h})\big), t_2 \cdots t_n\Big\rangle \quad \textit{if } \mathcal{S}_{fin} \in \{\infty\},\ \mathcal{R} = \mathcal{R}_1{}^{\downarrow} \\
\quad \textit{where } \langle t_2 \cdots t_n\rangle = \mathcal{M}_{\langle\rangle}\Big(t_{equ_t}, \mathcal{R}, \Big(\Delta' \setminus \big(\Delta'(t_{equ_h}) \cup \Delta'_h\big)\Big)\Big) \\
\quad \textit{s.t. } t_{equ} = t_{equ_h} \vee t_{equ_t} \textit{ and } \Delta'(t_{equ_h}) = (t_{equ_h}, t'_{equ_h}, \mathcal{R}_1, \mathcal{S}'_{fin}, \Delta'_h) \\
\mathcal{M}\big(\Delta'(t_{equ})\big)^{?} \quad \textit{if } \mathcal{S}_{fin} \in \{sol\} \textit{ and } \mathcal{R} = \mathcal{R}_1{}^{?} \textit{ for some } \mathcal{R}_1 \\
\mathcal{M}\big(\Delta'(t_{equ})\big)^{\leftarrow} \quad \textit{if } \mathcal{S}_{fin} \in \{L\} \\
\mathcal{M}\big(\Delta'(t_{equ})\big)^{\rightarrow} \quad \textit{if } \mathcal{S}_{fin} \in \{R\} \\
t \quad \textit{otherwise}
\end{cases}$$

$$\mathcal{M}_{\{\}}\big(t, \mathcal{R}, \Delta\big) := \begin{cases} \Big\{\mathcal{M}\big(\Delta(t)\big)\Big\} \quad \textit{if } t \in \mathcal{Dom}(\Delta) \\ \Big\{\mathcal{M}\big(\Delta(t_h)\big), t_2 \cdots t_m\Big\} \quad \textit{if } t = t_h \wedge t_t \textit{ s.t. } \Delta(t_h) = (t_h, t_{h_{equ}}, \mathcal{R}_1, \mathcal{S}_{fin_h}, \Delta'_h) \\ \quad \textit{where } \mathcal{R} = {\mathcal{R}_1}^*,\ \{t_2 \cdots t_m\} = \mathcal{M}_{\{\}}\Big(t_t, \mathcal{R}, \big(\Delta \setminus \Delta(t_h) \cup \Delta'_h\big)\Big) \\ \{t\} \quad \textit{otherwise} \end{cases}$$

$$\mathcal{M}_{[]}\big(t, \mathcal{R}, \Delta\big) := \begin{cases} \Big[\mathcal{M}\big(\Delta(t)\big)\Big] \quad \textit{if } t \in \mathcal{Dom}(\Delta) \\ \Big[\mathcal{M}\big(\Delta(t_h)\big), t_2 \cdots t_m\Big] \quad \textit{if } t = t_h \wedge t_t \textit{ s.t. } \Delta(t_h) = (t_h, t_{h_{equ}}, \mathcal{R}_1, \mathcal{S}_{fin_h}, \Delta'_h) \\ \quad \textit{where } \mathcal{R} = {\mathcal{R}_1}^+,\ [t_2 \cdots t_m] = \mathcal{M}_{[]}\Big(t_t, \mathcal{R}, \big(\Delta \setminus \Delta(t_h) \cup \Delta'_h\big)\Big) \\ [t] \quad \textit{otherwise} \end{cases}$$

$$\mathcal{M}_{\langle\rangle}\big(t, \mathcal{R}, \Delta\big) := \begin{cases} \Big\langle\mathcal{M}\big(\Delta(t)\big)\Big\rangle \quad \textit{if } t \in \mathcal{Dom}(\Delta) \\ \Big\langle\mathcal{M}\big(\Delta(t_h)\big), t_2 \cdots t_m\Big\rangle \quad \textit{if } t = t_h \vee t_t \textit{ s.t. } \Delta(t_h) = (t_h, t_{h_{equ}}, \mathcal{R}_1, \mathcal{S}_{fin_h}, \Delta'_h) \\ \quad \textit{where } \mathcal{R} = {\mathcal{R}_1}^{\downarrow},\ \langle t_2 \cdots t_m\rangle = \mathcal{M}_{\langle\rangle}\Big(t_t, \mathcal{R}, \big(\Delta \setminus \Delta(t_h) \cup \Delta'_h\big)\Big) \\ \langle t\rangle \quad \textit{otherwise} \end{cases}$$

Property 4.1

$$\mathcal{M}\big(\Delta(t)\big) \simeq t \textit{ s.t. } \Delta \vdash t \triangleright \mathcal{R}.$$

Proof. *Proof is induction on derivation of* $\Delta \vdash t \triangleright \mathcal{R}$. □

Lemma 4.2 *completeness on extended matching.*

$$\Gamma \vdash \mathcal{M}\big(\Delta(t)\big) : \mathcal{R} \ \textit{ for some } \Gamma, \ \textit{where } \Delta \vdash t \triangleright \mathcal{R}.$$

Proof. *Proof is shown in Section 7.*

5 Implementation

In this section, we refine our logic more suitably to implementation. Specifically, termination Property is necessary for machine computation. And based on the revised logic called as Algorithmic form, which has termination Property, we implement our logic on machine.

Definition 5.1 (intuitionistic equivalence for implementation) *We define the set of equivalent terms of t as $\mathcal{E}qu(t)$, on following rules.*

$$t \in \mathcal{E}qu(t)$$

$$\begin{array}{ll} \mathcal{A}ssoc_{\wedge}(t_1,t_2) \subseteq \mathcal{E}qu(t) & \textit{if } t = t_1 \wedge t_2 \textit{ for some } t_1,t_2 \\ \mathcal{A}ssoc_{\vee}(t_1,t_2) \subseteq \mathcal{E}qu(t) & \textit{if } t = t_1 \vee t_2 \textit{ for some } t_1,t_2 \end{array}$$

$$t_1 \in \mathcal{E}qu\big(\{t_1\}\big) \quad t_1 \in \mathcal{E}qu\big([t_1]\big) \quad t_1 \in \mathcal{E}qu\big(\langle t_1\rangle\big)$$

$$\begin{array}{ll} t_1 \wedge \{t_2,\cdots t_n\} \in \mathcal{E}qu\big(\{t_1,t_2,\cdots t_n\}\big) & t_1 \wedge [t_2,\cdots t_n] \in \mathcal{E}qu\big([t_1,t_2,\cdots t_n]\big) \\ & t_1 \vee \langle t_2,\cdots t_n\rangle \in \mathcal{E}qu\big(\langle t_1,t_2,\cdots t_n\rangle\big) \end{array}$$

$$\begin{array}{llll} \{\} \wedge t \in \mathcal{E}qu(t) & \textit{if } t \neq \{\} \wedge t_1 \textit{ for some } t_1. & \circ \wedge t \in \mathcal{E}qu(t) & \textit{if } t \neq \circ \wedge t_1 \textit{ for some } t_1. \\ t \wedge \{\} \in \mathcal{E}qu(t) & \textit{if } t \neq t_1 \wedge \{\} \textit{ for some } t_1. & t \wedge \circ \in \mathcal{E}qu(t) & \textit{if } t \neq t_1 \wedge \circ \textit{ for some } t_1. \end{array}$$

$$\begin{array}{llll} \{\} \vee t \in \mathcal{E}qu(t) & \textit{if } t \neq \{\} \vee t_1 \textit{ for some } t_1. & \circ \vee t \in \mathcal{E}qu(t) & \textit{if } t \neq \circ \vee t_1 \textit{ for some } t_1. \\ t \vee \{\} \in \mathcal{E}qu(t) & \textit{if } t \neq t_1 \vee \{\} \textit{ for some } t_1. & t \vee \circ \in \mathcal{E}qu(t) & \textit{if } t \neq t_1 \vee \circ \textit{ for some } t_1. \end{array}$$

$$t_1 \in \mathcal{E}qu\big(\{\} \wedge t_1\big) \quad t_1 \in \mathcal{E}qu\big(t_1 \wedge \{\}\big) \qquad t_1 \in \mathcal{E}qu\big(\circ \wedge t_1\big) \quad t_1 \in \mathcal{E}qu\big(t_1 \wedge \circ\big)$$

$$t_1 \in \mathcal{E}qu\big(\{\} \vee t_1\big) \quad t_1 \in \mathcal{E}qu\big(t_1 \vee \{\}\big) \qquad t_1 \in \mathcal{E}qu\big(\circ \vee t_1\big) \quad t_1 \in \mathcal{E}qu\big(t_1 \vee \circ\big)$$

$$t_1 \in \mathcal{E}qu({t_1}^{?}) \quad t_1 \in \mathcal{E}qu({t_1}^{\leftarrow}) \quad t_1 \in \mathcal{E}qu({t_1}^{\rightarrow})$$

where $\mathcal{A}ssoc_{\wedge}(t_1,t_2)$, $\mathcal{A}ssoc_{\vee}(t_1,t_2)$ and related functions are as follows.

$$\mathcal{A}ssoc_{\wedge}(t_1,t_2) \overset{\text{def}}{:=} \mathcal{A}ssoc_{\wedge}(t_1,t_2,\mathcal{L}_2\mathcal{R}) \cup \mathcal{A}ssoc_{\wedge}(t_1,t_2,\mathcal{R}_2\mathcal{L})$$

$$\mathcal{A}ssoc_{\wedge}(t_1,t_2,\mathcal{L}_2\mathcal{R}) \overset{\text{def}}{:=} \begin{cases} \{t_1 \wedge t_2\} & \textit{if } t'_1 \wedge t'_2 = t_1 \wedge t_2 \textit{ where } t'_1 \wedge t'_2 = \textit{assoc}_{\textit{cas}}\mathcal{R}(t_1,t_2) \\ \{t'_1 \wedge t'_2\} \cup \mathcal{A}ssoc_{\wedge}(t'_1,t'_2,\mathcal{L}_2\mathcal{R}) & \textit{otherwise} \end{cases}$$

$$\mathcal{A}ssoc_{\wedge}(t_1,t_2,\mathcal{R}_2\mathcal{L}) \overset{\text{def}}{:=} \begin{cases} \{t_1 \wedge t_2\} & \textit{if } t'_1 \wedge t'_2 = t_1 \wedge t_2 \textit{ where } t'_1 \wedge t'_2 = \textit{assoc}_{\textit{cas}}\mathcal{L}(t_1,t_2) \\ \{t'_1 \wedge t'_2\} \cup \mathcal{A}ssoc_{\wedge}(t'_1,t'_2,\mathcal{R}_2\mathcal{L}) & \textit{otherwise} \end{cases}$$

$$\textit{assoc}_{\textit{cas}}\mathcal{R}(t_1,t_2) \overset{\text{def}}{:=} \begin{cases} t_1 \wedge t_2 \quad \textit{if } \textit{una}(t_1) \textit{ or } t_1 = t_{1l} \vee t_{1r} \\ t_{1l} \wedge (t_{1r} \wedge t_2) \quad \textit{if } t_1 = t_{1l} \wedge t_{1r} \textit{ s.t. } \textit{una}(t_{1r}) \textit{ or } t_{1r} = t_{1r_l} \vee t_{1r_r} \\ (t_{1l} \wedge t'_{1r_l}) \wedge t'_{1r_r} \quad \textit{if } t_1 = t_{1l} \wedge t_{1r} \textit{ s.t. } t_{1r} = t_{1r_l} \wedge t_{1r_r} \\ \quad \textit{where } t'_{1r_l} \wedge t'_{1r_r} = \textit{assoc}_{\textit{cas}}\mathcal{R}(t_{1r_l} \wedge t_{1r_r}, t_2) \\ t_1 \wedge t_2 \quad \textit{otherwise} \end{cases}$$

$$\mathit{assoc}_{cas}\mathcal{L}(t_1, t_2) \overset{\text{def}}{:=} \begin{cases} t_1 \wedge t_2 \quad \textit{if } \mathit{una}(t_2) \textit{ or } t_2 = t_{2l} \vee t_{2r} \\ (t_1 \wedge t_{2l}) \wedge t_{2r} \quad \textit{if } t_2 = t_{2l} \wedge t_{2r} \textit{ s.t. } \mathit{una}(t_{2l}) \textit{ or } t_{2l} = t_{2l_l} \vee t_{2l_r} \\ t'_{2l_l} \wedge (t'_{2l_r} \wedge t_{2r}) \quad \textit{if } t_2 = t_{2l} \wedge t_{2r} \textit{ s.t. } t_{2l} = t_{2l_l} \wedge t_{2l_r} \\ \quad \textit{where } t'_{2l_l} \wedge t'_{2l_r} = \mathit{assoc}_{cas}\mathcal{L}(t_1, t_{2l_l} \wedge t_{2l_r}) \\ t_1 \wedge t_2 \quad \textit{otherwise} \end{cases}$$

$$\mathcal{A}\mathit{ssoc}_{\vee}(t_1, t_2) \overset{\text{def}}{:=} \mathcal{A}\mathit{ssoc}_{\vee}(t_1, t_2, \mathcal{L}_2\mathcal{R}) \cup \mathcal{A}\mathit{ssoc}_{\vee}(t_1, t_2, \mathcal{R}_2\mathcal{L})$$

$$\mathcal{A}\mathit{ssoc}_{\vee}(t_1, t_2, \mathcal{L}_2\mathcal{R}) \overset{\text{def}}{:=} \begin{cases} \{t_1 \vee t_2\} \quad \textit{if } t'_1 \vee t'_2 = t_1 \vee t_2 \textit{ where } t'_1 \vee t'_2 = \mathit{assoc}_{par}\mathcal{R}(t_1, t_2) \\ \{t'_1 \vee t'_2\} \cup \mathcal{A}\mathit{ssoc}_{\vee}(t'_1, t'_2, \mathcal{L}_2\mathcal{R}) \quad \textit{otherwise} \end{cases}$$

$$\mathcal{A}\mathit{ssoc}_{\vee}(t_1, t_2, \mathcal{R}_2\mathcal{L}) \overset{\text{def}}{:=} \begin{cases} \{t_1 \vee t_2\} \quad \textit{if } t'_1 \vee t'_2 = t_1 \vee t_2 \textit{ where } t'_1 \vee t'_2 = \mathit{assoc}_{par}\mathcal{L}(t_1, t_2) \\ \{t'_1 \vee t'_2\} \cup \mathcal{A}\mathit{ssoc}_{\vee}(t'_1, t'_2, \mathcal{R}_2\mathcal{L}) \quad \textit{otherwise} \end{cases}$$

$$\mathit{assoc}_{par}\mathcal{R}(t_1, t_2) \overset{\text{def}}{:=} \begin{cases} t_1 \vee t_2 \quad \textit{if } \mathit{una}(t_1) \textit{ or } t_1 = t_{1l} \wedge t_{1r} \\ t_{1l} \vee (t_{1r} \vee t_2) \quad \textit{if } t_1 = t_{1l} \vee t_{1r} \textit{ s.t. } \mathit{una}(t_{1r}) \textit{ or } t_{1r} = t_{1r_l} \wedge t_{1r_r} \\ (t_{1l} \vee t'_{1r_l}) \vee t'_{1r_r} \quad \textit{if } t_1 = t_{1l} \vee t_{1r} \textit{ s.t. } t_{1r} = t_{1r_l} \vee t_{1r_r} \\ \quad \textit{where } t'_{1r_l} \vee t'_{1r_r} = \mathit{assoc}_{par}\mathcal{R}(t_{1r_l} \vee t_{1r_r}, t_2) \\ t_1 \vee t_2 \quad \textit{otherwise} \end{cases}$$

$$\mathit{assoc}_{par}\mathcal{L}(t_1, t_2) \overset{\text{def}}{:=} \begin{cases} t_1 \vee t_2 \quad \textit{if } \mathit{una}(t_2) \textit{ or } t_2 = t_{2l} \wedge t_{2r} \\ (t_1 \vee t_{2l}) \vee t_{2r} \quad \textit{if } t_2 = t_{2l} \vee t_{2r} \textit{ s.t. } \mathit{una}(t_{2l}) \textit{ or } t_{2l} = t_{2l_l} \wedge t_{2l_r} \\ t'_{2l_l} \vee (t'_{2l_r} \vee t_{2r}) \quad \textit{if } t_2 = t_{2l} \vee t_{2r} \textit{ s.t. } t_{2l} = t_{2l_l} \vee t_{2l_r} \\ \quad \textit{where } t'_{2l_l} \vee t'_{2l_r} = \mathit{assoc}_{par}\mathcal{L}(t_1, t_{2l_l} \vee t_{2l_r}) \\ t_1 \vee t_2 \quad \textit{otherwise} \end{cases}$$

$$\mathit{una}(t) \overset{\text{def}}{:=} \begin{cases} \textit{true} \quad \textit{if} \begin{cases} t = t_0 \textit{ s.t. } (t_0, \mathcal{R}_0) \in \Gamma_{gnd} \\ t = t_1 \textit{ s.t. } (t_1, \mathcal{R}_1) \in \Gamma_{def} \\ t \in \{\{\}, \circ\} \\ t = \{t_1, \cdots t_n\} \textit{ s.t. } n \geq 1 \\ t = [t_1, \cdots t_n] \textit{ s.t. } n \geq 1 \\ t = \langle t_1, \cdots t_n \rangle \textit{ s.t. } n \geq 1 \\ t = t_1{}^{?} \\ t = t_1{}^{\leftarrow} \\ t = t_1{}^{\rightarrow} \end{cases} \\ \textit{false} \quad \textit{otherwise} \end{cases}$$

Property 5.1

$$t'_1 \wedge t'_2 \simeq t_1 \wedge t_2 \textit{ where } t'_1 \wedge t'_2 = \mathit{assoc}_{cas}\mathcal{R}(t_1, t_2).$$

Proof. *Proof is simple structual induction on t_1 of (t_1, t_2).*

From def. of $\mathit{assoc}_{cas}\mathcal{R}(t_1, t_2)$, Here are the only 2 cases to be considered. For one case, which $t_1 = t_{1l} \wedge t_{1r}$ s.t. t_{1r} satisfies $\mathit{una}(t_{1r})$ or t_{1r} has the form of $t_{1r} = t_{1r_l} \vee t_{1r_r}$, it's trivial from def. of $\mathit{assoc}_{cas}\mathcal{R}(t_1, t_2)$ with a derivation as below.

$$\left(\mathit{assoc}_{cas}\mathcal{R}(t_1,t_2) = t_{1l} \wedge (t_{1r} \wedge t_2)\right)\dfrac{\dfrac{(\mathit{Symmetry})\dfrac{(\mathit{Identity})\dfrac{\overline{(t_{1l} \wedge t_{1r}) \wedge t_2 \equiv (t_{1l} \wedge t_{1r}) \wedge t_2}}{\dfrac{(t_{1l} \wedge t_{1r}) \wedge t_2 \simeq (t_{1l} \wedge t_{1r}) \wedge t_2}{(t_{1l} \wedge t_{1r}) \wedge t_2 \simeq t_{1l} \wedge (t_{1r} \wedge t_2)}\,(\mathit{Associativity\text{-}Cas}_{\mathcal{R}})}}{t_{1l} \wedge (t_{1r} \wedge t_2) \simeq (t_{1l} \wedge t_{1r}) \wedge t_2}}{t_{1l} \wedge (t_{1r} \wedge t_2) \simeq t_1 \wedge t_2}\,(t_1 = t_{1l} \wedge t_{1r})}{\mathit{assoc}_{cas}\mathcal{R}(t_1,t_2) \simeq t_1 \wedge t_2}$$

For another case as $t_1 = t_{1l} \wedge (t_{1r_l} \wedge t_{1r_r})$. By induction hypothesis we suppose $t'_{1r_l} \wedge t'_{1r_r} \simeq (t_{1r_l} \wedge t_{1r_r}) \wedge t_2 = t_{1r} \wedge t_2$ where $t'_{1r_l} \wedge t'_{1r_r} = \mathit{assoc}_{cas}\mathcal{R}(t_{1r_l} \wedge t_{1r_r}, t_2) = \mathit{assoc}_{cas}\mathcal{R}(t_{1r}, t_2)$ for some t'_{1r_l} and t'_{1r_r}. Consequently we find a deduction for $\mathit{assoc}_{cas}\mathcal{R}(t_1, t_2) \simeq t_1 \wedge t_2$ as follow.

$$(\mathit{Symmetry})\dfrac{(\mathit{Identity})\dfrac{\overline{(t_{1l} \wedge t_{1r}) \wedge t_2 \equiv (t_{1l} \wedge t_{1r}) \wedge t_2}}{\dfrac{(t_{1l} \wedge t_{1r}) \wedge t_2 \simeq (t_{1l} \wedge t_{1r}) \wedge t_2}{(t_{1l} \wedge t_{1r}) \wedge t_2 \simeq t_{1l} \wedge (t_{1r} \wedge t_2)}\,(\mathit{Associativity\text{-}Cas}_{\mathcal{R}})}}{t_{1l} \wedge (t_{1r} \wedge t_2) \simeq (t_{1l} \wedge t_{1r}) \wedge t_2}$$

$$\dfrac{\dfrac{t'_{1r_l} \wedge t'_{1r_r} \simeq t_{1r} \wedge t_2 \ \textit{by I.H.}}{t_{1l} \wedge \left(t'_{1r_l} \wedge t'_{1r_r}\right) \simeq t_{1l} \wedge (t_{1r} \wedge t_2)}\,(\mathit{Transparency\text{-}Cas}) \qquad t_{1l} \wedge (t_{1r} \wedge t_2) \simeq (t_{1l} \wedge t_{1r}) \wedge t_2}{(\mathit{Symmetry})\dfrac{t_{1l} \wedge \left(t'_{1r_l} \wedge t'_{1r_r}\right) \simeq (t_{1l} \wedge t_{1r}) \wedge t_2 \ \textit{from } (\mathit{Transitivity})}{(t_1 = t_{1l} \wedge t_{1r})\dfrac{\dfrac{(t_{1l} \wedge t_{1r}) \wedge t_2 \simeq t_{1l} \wedge \left(t'_{1r_l} \wedge t'_{1r_r}\right)}{(t_{1l} \wedge t_{1r}) \wedge t_2 \simeq \left(t_{1l} \wedge t'_{1r_l}\right) \wedge t'_{1r_r}}\,(\mathit{Associativity\text{-}Cas}_{\mathcal{L}})}{(\mathit{Symmetry})\dfrac{t_1 \wedge t_2 \simeq \left(t_{1l} \wedge t'_{1r_l}\right) \wedge t'_{1r_r}}{\dfrac{\left(t_{1l} \wedge t'_{1r_l}\right) \wedge t'_{1r_r} \simeq t_1 \wedge t_2}{\mathit{assoc}_{cas}\mathcal{R}(t_1,t_2) \simeq t_1 \wedge t_2}\,\left(\mathit{assoc}_{cas}\mathcal{R}(t_1,t_2) = \left(t_{1l} \wedge t'_{1r_l}\right) \wedge t'_{1r_r}\right)}}}}$$

□

Property 5.2

$$\mathit{size}(t_1) > \mathit{size}(t_1') \;\; \textit{if}\, t_1 \wedge t_2 \neq t_1' \wedge t_2',\; \textit{where}\; t_1' \wedge t_2' = \mathit{assoc}_{cas}\mathcal{R}(t_1, t_2).$$

Proof. *Proof is simple induction on structure of* t_1 *according to def. of* $\mathit{assoc}_{cas}\mathcal{R}(t_1, t_2)$.

We are considering for each case of the structure on (t_1, t_2). *For the case that* t_1 *satisfies* $\mathit{una}(t_1)$ *or has the form of* $t_1 = t_{1l} \vee t_{1r}$ *for some* t_{1l} *and* t_{1r}, *it's obvious for* $t_1 \wedge t_2 = \mathit{assoc}_{cas}\mathcal{R}(t_1, t_2)$.

Then, we consider the case that $t_1 = t_{1l} \wedge t_{1r}$ *where* t_{1r} *satisfies* $\mathit{una}(t_{1r})$ *or has the form of* $t_{1r} = t_{1r_l} \vee t_{1r_r}$. *It's trivial as* $\mathit{size}(t_1) = \mathit{size}(t_{1l} \wedge t_{1r}) > \mathit{size}(t_{1l})$ *where* $t_{1l} \wedge (t_{1r} \wedge t_2) = \mathit{assoc}_{cas}\mathcal{R}(t_1, t_2)$.

Finally, for the remaining case that $t_1 = t_{1l} \wedge t_{1r}$ *s.t.* $t_{1r} = t_{1r_l} \wedge t_{1r_r}$ *to be considered, we suppose* $t'_{1r_l} \wedge t'_{1r_r} = \mathit{assoc}_{cas}\mathcal{R}(t_{1r_l} \wedge t_{1r_r}, t_2) = \mathit{assoc}_{cas}\mathcal{R}(t_{1r}, t_2)$. *For the case of* $t'_{1r_l} \wedge t'_{1r_r} = t_{1r} \wedge t_2$, *which implies the facts of* $t'_{1r_l} = t_{1r}$ *and* $t'_{1r_r} = t_2$. *Therefore prerequisite isn't satisfied, since* $\mathit{assoc}_{cas}\mathcal{R}(t_1, t_2) = (t_{1l} \wedge t'_{1r_l}) \wedge t'_{1r_r} = (t_{1l} \wedge t_{1r}) \wedge t_2 = t_1 \wedge t_2$. *Otherwise, i.e.* $t'_{1r_l} \wedge t'_{1r_r} \neq t_{1r} \wedge t_2$, *we know* $\mathit{size}(t_{1r}) > \mathit{size}(t'_{1r_l})$ *by induction hypothesis, thus* $\mathit{size}(t_1) = \mathit{size}(t_{1l} \wedge t_{1r}) = \mathit{size}(t_{1l}) + \mathit{size}(t_{1r}) + 1 > \mathit{size}(t_{1l}) + \mathit{size}(t'_{1r_l}) + 1 = \mathit{size}(t_{1l} \wedge t'_{1r_l}) = \mathit{size}(t_1')$ *as desired.* $\square$

Property 5.3

$$t_1' \wedge t_2' \simeq t_1 \wedge t_2 \;\textit{where}\; t_1' \wedge t_2' = \mathit{assoc}_{cas}\mathcal{L}(t_1, t_2).$$

Proof. *Proof is simlar to previous Property 5.1 with following deductions.*

$$\big(\mathit{assoc}_{cas}\mathcal{L}(t_1,t_2) = (t_1 \wedge t_{2l}) \wedge t_{2r}\big)\, \dfrac{(\mathit{Symmetry})\,\dfrac{(\mathit{Identity})\,\dfrac{\dfrac{}{t_1 \wedge (t_{2l} \wedge t_{2r}) \equiv t_1 \wedge (t_{2l} \wedge t_{2r})}}{t_1 \wedge (t_{2l} \wedge t_{2r}) \simeq t_1 \wedge (t_{2l} \wedge t_{2r})}}{\dfrac{t_1 \wedge (t_{2l} \wedge t_{2r}) \simeq (t_1 \wedge t_{2l}) \wedge t_{2r}}{(t_1 \wedge t_{2l}) \wedge t_{2r} \simeq t_1 \wedge (t_{2l} \wedge t_{2r})}}\,(\mathit{Associativity\text{-}Cas}_{\mathcal{L}})}{\dfrac{(t_1 \wedge t_{2l}) \wedge t_{2r} \simeq t_1 \wedge t_2}{\mathit{assoc}_{cas}\mathcal{L}(t_1, t_2) \simeq t_1 \wedge t_2}}\,(t_2 = t_{2l} \wedge t_{2r})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{t_1 \wedge (t_{2l} \wedge t_{2r}) \equiv t_1 \wedge (t_{2l} \wedge t_{2r})}}{t_1 \wedge (t_{2l} \wedge t_{2r}) \simeq t_1 \wedge (t_{2l} \wedge t_{2r})}\,(\mathit{Identity})}{t_1 \wedge (t_{2l} \wedge t_{2r}) \simeq (t_1 \wedge t_{2l}) \wedge t_{2r}}\,(\mathit{Associativity\text{-}Cas}_{\mathcal{L}})}{(t_1 \wedge t_{2l}) \wedge t_{2r} \simeq t_1 \wedge (t_{2l} \wedge t_{2r})}\,(\mathit{Symmetry})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{t'_{2l_l} \wedge t'_{2l_r} \simeq t_1 \wedge t_{2l} \;\; \textit{by I.H.}}{\left(t'_{2l_l} \wedge t'_{2l_r}\right) \wedge t_{2r} \simeq (t_1 \wedge t_{2l}) \wedge t_{2r}}\,(\mathit{Transparency\text{-}Cas}) \qquad (t_1 \wedge t_{2l}) \wedge t_{2r} \simeq t_1 \wedge (t_{2l} \wedge t_{2r})}{\left(t'_{2l_l} \wedge t'_{2l_r}\right) \wedge t_{2r} \simeq t_1 \wedge (t_{2l} \wedge t_{2r}) \;\; \textit{from } (\mathit{Transitivity})}}{t_1 \wedge (t_{2l} \wedge t_{2r}) \simeq \left(t'_{2l_l} \wedge t'_{2l_r}\right) \wedge t_{2r}}\,(\mathit{Symmetry})}{t_1 \wedge (t_{2l} \wedge t_{2r}) \simeq t'_{2l_l} \wedge \left(t'_{2l_r} \wedge t_{2r}\right)}\,(\mathit{Associativity\text{-}Cas}_{\mathcal{R}})}{t_1 \wedge t_2 \simeq t'_{2l_l} \wedge \left(t'_{2l_r} \wedge t_{2r}\right)}\,(t_2 = t_{2l} \wedge t_{2r})}{t'_{2l_l} \wedge \left(t'_{2l_r} \wedge t_{2r}\right) \simeq t_1 \wedge t_2}\,(\mathit{Symmetry})}{\mathit{assoc}_{cas}\mathcal{L}(t_1, t_2) \simeq t_1 \wedge t_2}\,\left(\mathit{assoc}_{cas}\mathcal{L}(t_1, t_2) = t'_{2l_l} \wedge \left(t'_{2l_r} \wedge t_{2r}\right)\right)$$

□

Property 5.4

$$\mathit{size}(t_2) > \mathit{size}(t'_2) \;\; \textit{if } t_1 \wedge t_2 \neq t'_1 \wedge t'_2, \; \textit{where } t'_1 \wedge t'_2 = \mathit{assoc}_{cas}\mathcal{L}(t_1, t_2).$$

Proof. *Proof is similar to Property 5.2.* □

Property 5.5

$$t' \simeq t_1 \wedge t_2 \textit{ where } t' \in \mathcal{A}\mathit{ssoc}_{\wedge}(t_1, t_2, \mathcal{L}_2\mathcal{R})$$

Proof. *Proof is induction on structure of* (t_1, t_2) *with the ordering scheme as below.*

$$\begin{array}{ll} (t_1, t_2) > (t'_1, t'_2) & \textit{if } \mathit{size}(t_1) > \mathit{size}(t'_1) \\ (t_1, t_2) > (t'_1, t'_2) & \textit{if } \mathit{size}(t_1) = \mathit{size}(t'_1) \textit{ and } \mathit{size}(t_2) > \mathit{size}(t'_2) \\ (t_1, t_2) = (t'_1, t'_2) & \textit{otherwise} \end{array}$$

From def. of $\mathcal{A}\mathit{ssoc}_{\wedge}(t_1, t_2, \mathcal{L}_2\mathcal{R})$, *we suppose* $t'_1 \wedge t'_2 = t_1 \wedge t_2$ *where* $t'_1 \wedge t'_2 = \mathit{assoc}_{cas}\mathcal{R}(t_1, t_2)$. *In this case we know* $\mathcal{A}\mathit{ssoc}_{\wedge}(t_1, t_2, \mathcal{L}_2\mathcal{R}) = \{t_1 \wedge t_2\}$, *and it's trivial that* $t_1 \wedge t_2 \simeq t_1 \wedge t_2$ *from Identity. Otherwise, i.e.* $t'_1 \wedge t'_2 \neq t_1 \wedge t_2$, *we also know that* $\mathcal{A}\mathit{ssoc}_{\wedge}(t_1, t_2, \mathcal{L}_2\mathcal{R}) =$

$\{t'_1 \wedge t'_2\} \cup \mathcal{A}ssoc_{\wedge}(t'_1, t'_2, \mathcal{L}_2\mathcal{R})$ from def. of $\mathcal{A}ssoc_{\wedge}(t_1, t_2, \mathcal{L}_2\mathcal{R})$. From Property 5.1, $t'_1 \wedge t'_2 \simeq t_1 \wedge t_2$ where $t'_1 \wedge t'_2 = assoc_{cas}\mathcal{R}(t_1, t_2)$. We still know the fact of $size(t_1) > size(t'_1)$ from Property 5.2, thus for any $t'' \in \mathcal{A}ssoc_{\wedge}(t'_1, t'_2, \mathcal{L}_2\mathcal{R})$ satisfies $t'' \simeq t'_1 \wedge t'_2$ by induction hypothesis according to above ordering scheme. Accounting on the fact of $t'_1 \wedge t'_2 \simeq t_1 \wedge t_2$ as above, we can say that $t' \simeq t_1 \wedge t_2$ for any $t' \in \big(\{t'_1 \wedge t'_2\} \cup \mathcal{A}ssoc_{\wedge}(t'_1, t'_2, \mathcal{L}_2\mathcal{R})\big) = \mathcal{A}ssoc_{\wedge}(t_1, t_2, \mathcal{L}_2\mathcal{R})$ from Transitivity.

$\square$

Property 5.6

$$t' \simeq t_1 \wedge t_2 \text{ where } t' \in \mathcal{A}ssoc_{\wedge}(t_1, t_2, \mathcal{R}_2\mathcal{L})$$

Proof. *Proof is similar to Property 5.5 using Property 5.3, 5.4 with the ordering scheme as follow.*

$$\begin{array}{ll} (t_1, t_2) > (t'_1, t'_2) & \textit{if } size(t_2) > size(t'_2) \\ (t_1, t_2) > (t'_1, t'_2) & \textit{if } size(t_2) = size(t'_2) \textit{ and } size(t_1) > size(t'_1) \\ (t_1, t_2) = (t'_1, t'_2) & \textit{otherwise} \end{array}$$

$\square$

Lemma 5.1 *equivalency on gathering equivalents*

$$t \simeq t' \text{ where } t' \in \mathcal{E}qu(t)$$

Proof. *Proof is induction on structure of t according to def. of $\mathcal{E}qu(t)$.*

Case t_0 s.t. $(t_0, \mathcal{R}_0) \in \Gamma_{gnd}$: It's obvious that $\mathcal{E}qu(t_0) = \{t_0, \{\} \wedge t_0, t_0 \wedge \{\}, \{\} \vee t_0, t_0 \vee \{\}, \circ \wedge t_0, t_0 \wedge \circ, \circ \vee t_0, t_0 \vee \circ\}$ from def. of $\mathcal{E}qu(t)$, thus we can easily see its equivalence. $t_0 \simeq t_0$ is by Identity. For $\{\} \wedge t_0$, $t_0 \wedge \{\}$ and $\{\} \vee t_0$, $t_0 \vee \{\}$ are also trivial as follows.

$$(\textit{Identity})\,\dfrac{\dfrac{\overline{t_0 \equiv t_0}}{t_0 \simeq t_0}}{t_0 \simeq \{\} \wedge t_0}\,(\textit{PhonyH-Cas}_{\{\}}) \qquad (\textit{Identity})\,\dfrac{\dfrac{\overline{t_0 \equiv t_0}}{t_0 \simeq t_0}}{t_0 \simeq t_0 \wedge \{\}}\,(\textit{PhonyT-Cas}_{\{\}})$$

$$(\textit{Identity})\,\dfrac{\dfrac{\overline{t_0 \equiv t_0}}{t_0 \simeq t_0}}{t_0 \simeq \{\} \vee t_0}\,(\textit{PhonyH-Par}_{\{\}}) \qquad (\textit{Identity})\,\dfrac{\dfrac{\overline{t_0 \equiv t_0}}{t_0 \simeq t_0}}{t_0 \simeq t_0 \vee \{\}}\,(\textit{PhonyT-Par}_{\{\}})$$

For $\circ \wedge t_0$, $t_0 \wedge \circ$ and $\circ \vee t_0$, $t_0 \vee \circ$ are similar as follows.

$$(\textit{Identity})\,\dfrac{\dfrac{\overline{t_0 \equiv t_0}}{t_0 \simeq t_0}}{t_0 \simeq \circ \wedge t_0}\,(\textit{PhonyH-Cas}_{\circ}) \qquad (\textit{Identity})\,\dfrac{\dfrac{\overline{t_0 \equiv t_0}}{t_0 \simeq t_0}}{t_0 \simeq t_0 \wedge \circ}\,(\textit{PhonyT-Cas}_{\circ})$$

$$(\textit{Identity})\,\dfrac{\dfrac{\overline{t_0 \equiv t_0}}{t_0 \simeq t_0}}{t_0 \simeq \circ \vee t_0}\,(\textit{PhonyH-Par}_{\circ}) \qquad (\textit{Identity})\,\dfrac{\dfrac{\overline{t_0 \equiv t_0}}{t_0 \simeq t_0}}{t_0 \simeq t_0 \vee \circ}\,(\textit{PhonyT-Par}_{\circ})$$

And remaining cases on $\mathcal{E}qu$ are excluded from syntax directness of $t = t_0$.

Case t_1 s.t. $(t_1, \mathcal{R}_1) \in \Gamma_{def}$: Similar to the case of t_0 s.t. $(t_0, \mathcal{R}_0) \in \Gamma_{gnd}$.

Case $t_1 \wedge t_2$: $t_1 \wedge t_2 \simeq t_1 \wedge t_2$ s.t. $t_1 \wedge t_2 \in \mathcal{E}qu(t_1 \wedge t_2)$ is trivially from Identity. Next, we know that $t' \simeq t_1 \wedge t_2$ for any $t' \in \mathcal{A}ssoc_{\wedge}(t_1, t_2)$ where $\mathcal{A}ssoc_{\wedge}(t_1, t_2) = \big(\mathcal{A}ssoc_{\wedge}(t_1, t_2, \mathcal{L}_2\mathcal{R}) \cup \mathcal{A}ssoc_{\wedge}(t_1, t_2, \mathcal{R}_2\mathcal{L})\big) \subseteq \mathcal{E}qu(t_1 \wedge t_2)$, since $t'' \simeq t_1 \wedge t_2$ for both $t'' \in \mathcal{A}ssoc_{\wedge}(t_1, t_2, \mathcal{L}_2\mathcal{R})$ and $t'' \in \mathcal{A}ssoc_{\wedge}(t_1, t_2, \mathcal{R}_2\mathcal{L})$ from Property 5.5 and 5.6 respectively. For any cases of $t \in \{\{\} \wedge t_2, t_1 \wedge \{\}, \circ \wedge t_2, t_1 \wedge \circ, \}$, we can derive its equivalence as follows.

$$\dfrac{\dfrac{\overline{\{\} \wedge t_2 \equiv \{\} \wedge t_2}}{\{\} \wedge t_2 \simeq \{\} \wedge t_2}\,(\textit{Identity})}{\{\} \wedge t_2 \simeq t_2}\,(\textit{RevealH-Cas}_{\{\}}) \qquad \dfrac{\dfrac{\overline{t_1 \wedge \{\} \equiv t_1 \wedge \{\}}}{t_1 \wedge \{\} \simeq t_1 \wedge \{\}}\,(\textit{Identity})}{t_1 \wedge \{\} \simeq t_1}\,(\textit{RevealT-Cas}_{\{\}})$$

$$\dfrac{\dfrac{\overline{\circ \wedge t_2 \equiv \circ \wedge t_2}}{\circ \wedge t_2 \simeq \circ \wedge t_2}\,(\textit{Identity})}{\circ \wedge t_2 \simeq t_2}\,(\textit{RevealH-Cas}_{\circ}) \qquad \dfrac{\dfrac{\overline{t_1 \wedge \circ \equiv t_1 \wedge \circ}}{t_1 \wedge \circ \simeq t_1 \wedge \circ}\,(\textit{Identity})}{t_1 \wedge \circ \simeq t_1}\,(\textit{RevealT-Cas}_{\circ})$$

On accounting its syntax directness, remainders i.e. the term $t' \in \mathcal{E}qu(t_1 \wedge t_2)$ which equality s.t. $t' \simeq t_1 \wedge t_2$ we confirm using PhonyH-Cas$_{\{\}}$, PhonyT-Cas$_{\{\}}$, PhonyH-Cas$_{\circ}$, PhonyT-Cas$_{\circ}$, are similar to previous case.

Case $t_1 \vee t_2$: Similar to the case of $t = t_1 \wedge t_2$ with similar Properties.

Case $t = \{\}$: It's trivial that $t' \simeq \{\}$ where $t' \in \mathcal{E}qu\big(\{\}\big)$ using PhonyH-Cas$_{\{\}}$, PhonyT-Cas$_{\{\}}$, PhonyH-Par$_{\{\}}$, PhonyT-Par$_{\{\}}$, PhonyH-Cas$_{\circ}$, PhonyT-Cas$_{\circ}$, PhonyH-Par$_{\circ}$, PhonyT-Par$_{\circ}$ and Identity.

Case $t = \{t_1\}$: The only case to be particularly considered is as follow.

$$(\textit{Symmetry})\,\dfrac{\dfrac{(\textit{Identity})\,\dfrac{\overline{t_1 \equiv t_1}}{t_1 \simeq t_1}}{t_1 \simeq \{t_1\}}\,(\textit{Associativity-Cat0}_1)}{\{t_1\} \simeq t_1}$$

And remainders, i.e. the terms concerning PhonyH-Cas$_{\{\}}$, PhonyT-Cas$_{\{\}}$, PhonyH-Par$_{\{\}}$, PhonyT-Par$_{\{\}}$, PhonyH-Cas$_{\circ}$, PhonyT-Cas$_{\circ}$, PhonyH-Par$_{\circ}$, PhonyT-Par$_{\circ}$ and Identity for equivalency confirmation, are similar to the cases of above.

Case $t = \{t_1, t_2, \cdots t_n\}$ where $n \geq 2$: The only one to be particularly considered for this case is as follow.

$$(\textit{Symmetry})\,\dfrac{\dfrac{(\textit{Identity})\,\dfrac{\overline{t_1 \wedge \{t_2, \cdots t_n\} \equiv t_1 \wedge \{t_2, \cdots t_n\}}}{t_1 \wedge \{t_2, \cdots t_n\} \simeq t_1 \wedge \{t_2, \cdots t_n\}}}{t_1 \wedge \{t_2, \cdots t_n\} \simeq \{t_1, t_2, \cdots t_n\}}\,(\textit{AssociativityH-Cat0}_{\infty})}{\{t_1, t_2, \cdots t_n\} \simeq t_1 \wedge \{t_2, \cdots t_n\}}$$

Case $t = \circ$: This case is similar to the case of $t = \{\}$.

Case $t = [t_1]$: Similar to the case of $t = \{t_1\}$ with Associativity-Cat1$_1$.

Case $t = [t_1, t_2, \cdots t_n]$ where $n \geq 2$: Similar to the case of $t = \{t_1, t_2, \cdots t_n\}$ with AssociativityH-Cat1$_\infty$.

Case $t = \langle t_1 \rangle$: Similar to the case of $t = [t_1]$ with Associativity-Dup$_1$.

Case $t = \langle t_1, t_2, \cdots t_n \rangle$ where $n \geq 2$: Similar to the case of $t = [t_1, t_2, \cdots t_n]$ with AssociativityH-Dup$_\infty$.

Case $t = {t_1}^?$, $t = {t_1}^\leftarrow$, $t = {t_1}^\rightarrow$: The ones need paticular consideration are as follows.

$$\textit{(Symmetry)}\ \dfrac{\dfrac{\textit{(Identity)}\ \dfrac{\dfrac{}{t_0 \equiv t_0}}{t_0 \simeq t_0}}{t_0 \simeq {t_0}^?}\ \textit{(Optional)}}{{t_0}^? \simeq t_0} \qquad \textit{(Symmetry)}\ \dfrac{\dfrac{\textit{(Identity)}\ \dfrac{\dfrac{}{t_0 \equiv t_0}}{t_0 \simeq t_0}}{t_0 \simeq {t_0}^\leftarrow}\ \textit{(Alt-L)}}{{t_0}^\leftarrow \simeq t_0} \qquad \textit{(Symmetry)}\ \dfrac{\dfrac{\textit{(Identity)}\ \dfrac{\dfrac{}{t_0 \equiv t_0}}{t_0 \simeq t_0}}{t_0 \simeq {t_0}^\rightarrow}\ \textit{(Alt-R)}}{{t_0}^\rightarrow \simeq t_0}$$

And remaining cases are similar to above cases. □

Lemma 5.2 *termination on gathering equivalents*

$$\mathcal{E}qu(t) \textit{ has termination.}$$

Proof. *The only definitions on $\mathcal{E}qu$, which we should consider for termination, is $\mathcal{A}ssoc_\wedge(t_1, t_2)$ and $\mathcal{A}ssoc_\vee(t_1, t_2)$.*

For former, termination on both $assoc_{cas}\mathcal{R}(t_1, t_2)$ and $assoc_{cas}\mathcal{L}(t_1, t_2)$ are trivial by structual induction on t_1. Consequently, we are confirming the termination on $\mathcal{A}ssoc_\wedge(t_1, t_2, \mathcal{L}_2\mathcal{R})$. Accounting on above fact, i.e. $assoc_{cas}\mathcal{R}(t_1, t_2)$ has termination, it's obvious that $\mathcal{A}ssoc_\wedge(t_1, t_2, \mathcal{L}_2\mathcal{R})$ also has termination in the case of $t'_1 \wedge t'_2 = t_1 \wedge t_2$ where $t'_1 \wedge t'_2 = assoc_{cas}\mathcal{R}(t_1, t_2)$, from def. of $\mathcal{A}ssoc_\wedge(t_1, t_2, \mathcal{L}_2\mathcal{R})$.

Otherwise, i.e. $t'_1 \wedge t'_2 \neq t_1 \wedge t_2$, where we know that $size(t_1) > size(t'_1)$ from Property 5.2. Therefore we conclude that $\mathcal{A}ssoc_\wedge(t_1, t_2, \mathcal{L}_2\mathcal{R})$ has termination, since it contradicts the fact that t'_1 of $\mathcal{A}ssoc_\wedge(t'_1, t'_2, \mathcal{L}_2\mathcal{R})$ has finite and positve size.

We can find termination on $\mathcal{A}ssoc_{\wedge}(t_1, t_2, \mathcal{R}_2\mathcal{L})$ similarly, with Property 5.4 according to the ordering scheme over (t_1, t_2) as below.

$$\begin{array}{ll} (t_1,t_2) > (t'_1,t'_2) & \textit{if size}(t_2) > \textit{size}(t'_2) \\ (t_1,t_2) > (t'_1,t'_2) & \textit{if size}(t_2) = \textit{size}(t'_2) \textit{ and size}(t_1) > \textit{size}(t'_1) \\ (t_1,t_2) = (t'_1,t'_2) & \textit{otherwise} \end{array}$$

Note that, above ordering scheme is same as the one on Property 5.6.

Hence $\mathcal{A}ssoc_{\wedge}(t_1, t_2)$ has termination for $\mathcal{A}ssoc_{\wedge}(t_1, t_2) = \mathcal{A}ssoc_{\wedge}(t_1, t_2, \mathcal{L}_2\mathcal{R}) \cup \mathcal{A}ssoc_{\wedge}(t_1, t_2, \mathcal{R}_2\mathcal{L})$ from def. of $\mathcal{E}qu(t)$.

The latter one, i.e. $\mathcal{A}ssoc_{\vee}(t_1, t_2)$ is similar. □

Property 5.7

$$\textit{size}(t') \leq \textit{size}(t) + 2 \ \ \textit{where } t' \in \mathcal{E}qu(t)$$

Proof. *From def. of $\mathcal{E}qu(t)$, we focus on the only cases to be particularly considered, that are $\mathcal{A}ssoc_{\wedge}(t_1, t_2)$ and $\mathcal{A}ssoc_{\vee}(t_1, t_2)$ s.t. $t = t_1 \wedge t_2$ and $t = t_1 \vee t_2$ respectively.*

Here, we can say that both $assoc_{cas}\mathcal{R}(t_1, t_2)$ and $assoc_{cas}\mathcal{L}(t_1, t_2)$ satisfy the conditions of $\textit{size}(t'_1 \wedge t'_2) = \textit{size}(t_1 \wedge t_2)$ where $t'_1 \wedge t'_2 = assoc_{cas}\mathcal{R}(t_1, t_2)$ and $t'_1 \wedge t'_2 = assoc_{cas}\mathcal{L}(t_1, t_2)$ respectively, from induction hypothesis according to the reasoning as below.

Def. of $assoc_{cas}\mathcal{R}(t_1, t_2)$ has 2 cases to be considered. One is the case of $t_{1l} \wedge (t_{1r} \wedge t_2) = assoc_{cas}\mathcal{R}(t_1, t_2)$ s.t. $t_1 = t_{1l} \wedge t_{1r}$. It's trivial for $\textit{size}\big(t_{1l} \wedge (t_{1r} \wedge t_2)\big) = \textit{size}(t_{1l}) + \textit{size}(t_{1r} \wedge t_2) + 1 = \textit{size}(t_{1l}) + \big(\textit{size}(t_{1r}) + \textit{size}(t_2) + 1\big) + 1 = \big(\textit{size}(t_{1l}) + \textit{size}(t_{1r}) + 1\big) + \textit{size}(t_2) + 1 = \textit{size}(t_{1l} \wedge t_{1r}) + \textit{size}(t_2) + 1 = \textit{size}(t_1) + \textit{size}(t_2) + 1 = \textit{size}(t_1 \wedge t_2)$.

For another case, we suppose $(t_{1l} \wedge t'_{1r_l}) \wedge t'_{1r_r} = assoc_{cas}\mathcal{R}(t_1, t_2)$ where $t'_{1r_l} \wedge t'_{1r_r} = assoc_{cas}\mathcal{R}(t_{1r_l} \wedge t_{1r_r}, t_2) = assoc_{cas}\mathcal{R}(t_{1r}, t_2)$. Here we find $\textit{size}(t'_{1r_l}) + \textit{size}(t'_{1r_r}) = \textit{size}(t_{1r}) + \textit{size}(t_2)$ since $\textit{size}(t'_{1r_l} \wedge t'_{1r_r}) = \textit{size}(t_{1r} \wedge t_2)$ by applying induction hypothesis on $assoc_{cas}\mathcal{R}(t_{1r}, t_2)$ s.t. $t_1 = t_{1l} \wedge t_{1r}$. Therefore $\textit{size}\big((t_{1l} \wedge t'_{1r_l}) \wedge t'_{1r_r}\big) = \big(\textit{size}(t_{1l}) + \textit{size}(t'_{1r_l}) + 1\big) + \textit{size}(t'_{1r_r}) + 1 = \Big(\textit{size}(t_{1l}) + \big(\textit{size}(t'_{1r_l}) + \textit{size}(t'_{1r_r})\big) + 1\Big) + 1 = \Big(\textit{size}(t_{1l}) + \big(\textit{size}(t_{1r}) + \textit{size}(t_2)\big) + 1\Big) + 1 =$

$\big(size(t_{1l}) + size(t_{1r}) + 1\big) + size(t_2) + 1 = size(t_{1l} \wedge t_{1r}) + size(t_2) + 1 = size(t_1) + size(t_2) + 1 = size(t_1 \wedge t_2)$ *as desired. Similarly we also can see that for* $assoc_{cas}\mathcal{L}(t_1, t_2)$.

Then we examine for both cases according to whether $t'_1 \wedge t'_2 = t_1 \wedge t_2$ *or not, where* $t'_1 \wedge t'_2 = assoc_{cas}\mathcal{R}(t_1, t_2)$. *The former case is trivial for* $\{t_1 \wedge t_2\} = \mathcal{A}ssoc_\wedge(t_1, t_2, \mathcal{L}_2\mathcal{R})$, *where* $t_1 \wedge t_2 = assoc_{cas}\mathcal{R}(t_1, t_2)$. *And for latter case, we know* $size(t_1) > size(t'_1)$ *from Property 5.2, thus we can say that* $size(t') = size(t'_1 \wedge t'_2)$ *for any* $t' \in \mathcal{A}ssoc_\wedge(t'_1, t'_2, \mathcal{L}_2\mathcal{R})$ *from induction hypothesis. And we still know that* $size(t'_1 \wedge t'_2) = size(t_1 \wedge t_2)$ *where* $t'_1 \wedge t'_2 = assoc_{cas}\mathcal{R}(t_1, t_2)$ *from the fact revealed above.Therefore* $size(t') = size(t_1 \wedge t_2)$ *for any* $t' \in \mathcal{A}ssoc_\wedge(t_1, t_2, \mathcal{L}_2\mathcal{R})$ *since* $\mathcal{A}ssoc_\wedge(t_1, t_2, \mathcal{L}_2\mathcal{R}) = \{t'_1 \wedge t'_2\} \cup \mathcal{A}ssoc_\wedge(t'_1, t'_2, \mathcal{L}_2\mathcal{R})$, *where* $t'_1 \wedge t'_2 = assoc_{cas}\mathcal{R}(t_1, t_2)$ *s.t.* $size(t'_1 \wedge t'_2) = size(t_1 \wedge t_2)$. *We similarly can try it for* $\mathcal{A}ssoc_\wedge(t_1, t_2, \mathcal{R}_2\mathcal{L})$, *i.e.* $size(t') = size(t_1 \wedge t_2)$ *for any* $t' \in \mathcal{A}ssoc_\wedge(t_1, t_2, \mathcal{R}_2\mathcal{L})$, *with Property 5.4 and same ordering scheme as Property 5.6.*

Consequently, we conclude that $size(t') = size(t_1 \wedge t_2)$ *for any* $t' \in \big(\mathcal{A}ssoc_\wedge(t_1, t_2, \mathcal{L}_2\mathcal{R}) \cup \mathcal{A}ssoc_\wedge(t_1, t_2, \mathcal{R}_2\mathcal{L})\big) = \mathcal{A}ssoc_\wedge(t_1, t_2)$.

And according to def. of $\mathcal{E}qu(t)$, *the remaining cases of* $\mathcal{E}qu(t)$ *are limited to* $size(t) + 2$. $\square$

Definition 5.2 (implementation form of matching rules on polymorphic equivalent terms) *We modify the extended matching rules for implementation, which is also called as algorithmic form, as follows.*

$$\frac{\overline{t_0 \in \mathcal{E}qu(t) \quad (t_0, \mathcal{R}_0) \in \Gamma_{gnd}}}{\big\{(t, t_0, \mathcal{R}_0, gnd, \phi)\big\} \vdash t \blacktriangleright \mathcal{R}_0} \; \textit{(T-Atom0-impl)}$$

$$\frac{\overline{t_1 \in \mathcal{E}qu(t) \quad (t_1, \mathcal{R}_1) \in \Gamma_{def}}}{\big\{(t, t_1, \mathcal{R}_1, def, \phi)\big\} \vdash t \blacktriangleright \mathcal{R}_1} \; \textit{(T-Atom1-impl)}$$

$$\frac{\begin{array}{c}\Lambda_1 \vdash t_1 \blacktriangleright \mathcal{R}_1 \quad \Lambda_2 \vdash t_2 \blacktriangleright \mathcal{R}_2 \quad \mathcal{D}om(\Lambda_1) \cap \mathcal{D}om(\Lambda_2) = \phi \\ t_1 \wedge t_2 \in \mathcal{E}qu(t) \textit{ s.t. } t,\, t_1 \wedge t_2 \notin \mathcal{D}om(\Lambda_1 \cup \Lambda_2)\end{array}}{\Big\{\big(t, t_1 \wedge t_2, \mathcal{R}_1 \wedge \mathcal{R}_2, \wedge, (\Lambda_1 \cup \Lambda_2)\big)\Big\} \cup (\Lambda_1 \cup \Lambda_2) \vdash t \blacktriangleright \mathcal{R}_1 \wedge \mathcal{R}_2} \; \textit{(T-Cas-impl)}$$

$$\frac{\begin{array}{c}\Lambda_1 \vdash t_1 \blacktriangleright \mathcal{R}_1 \quad \Lambda_2 \vdash t_2 \blacktriangleright \mathcal{R}_2 \quad \mathcal{D}om(\Lambda_1) \cap \mathcal{D}om(\Lambda_2) = \phi \\ t_1 \vee t_2 \in \mathcal{E}qu(t) \textit{ s.t. } t,\, t_1 \vee t_2 \notin \mathcal{D}om(\Lambda_1 \cup \Lambda_2)\end{array}}{\Big\{\big(t, t_1 \vee t_2, \mathcal{R}_1 \vee \mathcal{R}_2, \vee, (\Lambda_1 \cup \Lambda_2)\big)\Big\} \cup (\Lambda_1 \cup \Lambda_2) \vdash t \blacktriangleright \mathcal{R}_1 \vee \mathcal{R}_2} \; \textit{(T-Par-impl)}$$

$$\left.\begin{array}{r}
\dfrac{\overline{\{\} \in \mathcal{E}qu(t) \quad \mathcal{R}_1 \textit{ is Any}}}{\Big\{\big(t, \{\}, \mathcal{R}_1^*, nil, \phi\big)\Big\} \vdash t \blacktriangleright {\mathcal{R}_1}^*} \quad (nil) \\[2em]
\dfrac{\Lambda' \vdash t' \blacktriangleright \mathcal{R}_1 \quad t' \in \mathcal{E}qu(t) \textit{ s.t. } t \notin \mathcal{D}om(\Lambda')}{\big\{(t, t', {\mathcal{R}_1}^*, sol, \Lambda')\big\} \cup \Lambda' \vdash t \blacktriangleright {\mathcal{R}_1}^*} \quad (sol) \\[2em]
\dfrac{\begin{array}{c}\Lambda_h \vdash t_h \blacktriangleright \mathcal{R}_1 \quad \Lambda_t \vdash t_t \blacktriangleright {\mathcal{R}_1}^* \quad \mathcal{D}om(\Lambda_h) \cap \mathcal{D}om(\Lambda_t) = \phi \\ t' \in \mathcal{E}qu(t) \textit{ s.t. } t' = \{\} \wedge (t_h \wedge t_t) \;\; \textit{or} \;\; \circ \wedge (t_h \wedge t_t) \;\; \textit{or} \;\; t_h \wedge t_t \\ \left\{\begin{array}{l}\{t,\, t_h \wedge t'_t\} \cap \mathcal{D}om(\Lambda_h \cup \Lambda_t) = \phi \quad \textit{if } \mathcal{S}_{fin_t} \in \{nil,\, sol\} \\ \{t,\, t_h \wedge t'_t,\, t'_t\} \cap \mathcal{D}om(\Lambda_h \cup \Lambda_t) = \phi \quad \textit{otherwise}\end{array}\right.\end{array}}{\begin{array}{c}\Big\{\big(t, t_h \wedge t'_t, {\mathcal{R}_1}^*, \infty, (\Lambda_h \cup \Lambda'_t)\big)\Big\} \cup (\Lambda_h \cup \Lambda'_t) \vdash t \blacktriangleright {\mathcal{R}_1}^* \\ \textit{where } \Lambda_t(t_t) = (t_t, t'_t, {\mathcal{R}_1}^*, \mathcal{S}_{fin_t}, \Lambda'_t) \textit{ s.t. } \mathcal{S}_{fin_t} \in \{sol, \infty\},\; t'_t \neq \{\}\end{array}} \quad (\infty)
\end{array}\right\} \textit{(T-Cat0-impl)}$$

$$
\left.\begin{array}{c}
\dfrac{\Lambda' \vdash t' \blacktriangleright \mathcal{R}_1 \quad t' \in \mathcal{E}qu(t)\ \textit{s.t.}\ t \notin \mathcal{D}om(\Lambda')}{\big\{(t, t', {\mathcal{R}_1}^{+}, sol, \Lambda')\big\} \cup \Lambda' \vdash t \blacktriangleright {\mathcal{R}_1}^{+}}\ (sol) \\[3em]
\dfrac{\begin{array}{c}
\Lambda_h \vdash t_h \blacktriangleright \mathcal{R}_1 \quad \Lambda_t \vdash t_t \blacktriangleright {\mathcal{R}_1}^{+} \quad \mathcal{D}om(\Lambda_h) \cap \mathcal{D}om(\Lambda_t) = \phi \\
t' \in \mathcal{E}qu(t)\ \textit{s.t.}\ t' = \{\} \wedge (t_h \wedge t_t)\ \ \textit{or}\ \ \circ \wedge (t_h \wedge t_t) \\
\qquad\qquad (t_h \wedge t_t) \wedge \{\}\ \ \textit{or}\ \ (t_h \wedge t_t) \wedge \circ\ \ \textit{or}\ \ t_h \wedge t_t \\
\begin{cases} \{t,\ t_h \wedge t'_t\} \cap \mathcal{D}om(\Lambda_h \cup \Lambda_t) = \phi & \textit{if}\ \mathcal{S}_{fin_t} = sol \\ \{t,\ t_h \wedge t'_t,\ t'_t\} \cap \mathcal{D}om(\Lambda_h \cup \Lambda_t) = \phi & \textit{otherwise} \end{cases}
\end{array}}{\begin{array}{c}
\Big\{\big(t, t_h \wedge t'_t, {\mathcal{R}_1}^{+}, \infty, (\Lambda_h \cup \Lambda'_t)\big)\Big\} \cup (\Lambda_h \cup \Lambda'_t) \vdash t \blacktriangleright {\mathcal{R}_1}^{+} \\
\textit{where}\ \Lambda_t(t_t) = (t_t, t'_t, {\mathcal{R}_1}^{+}, \mathcal{S}_{fin_t}, \Lambda'_t)\ \textit{s.t.}\ \mathcal{S}_{fin_t} \in \{sol, \infty\}
\end{array}}\ (\infty)
\end{array}\right\}\ \textit{(T-Cat1-impl)}
$$

$$
\left.\begin{array}{c}
\dfrac{\Lambda' \vdash t' \blacktriangleright \mathcal{R}_1 \quad t' \in \mathcal{E}qu(t)\ \textit{s.t.}\ t \notin \mathcal{D}om(\Lambda')}{\big\{(t, t', {\mathcal{R}_1}^{\downarrow}, sol, \Lambda')\big\} \cup \Lambda' \vdash t \blacktriangleright {\mathcal{R}_1}^{\downarrow}}\ (sol) \\[3em]
\dfrac{\begin{array}{c}
\Lambda_h \vdash t_h \blacktriangleright \mathcal{R}_1 \quad \Lambda_t \vdash t_t \blacktriangleright {\mathcal{R}_1}^{\downarrow} \quad \mathcal{D}om(\Lambda_h) \cap \mathcal{D}om(\Lambda_t) = \phi \\
t' \in \mathcal{E}qu(t)\ \textit{s.t.}\ t' = \{\} \vee (t_h \vee t_t)\ \ \textit{or}\ \ \circ \vee (t_h \vee t_t) \\
\qquad\qquad (t_h \vee t_t) \vee \{\}\ \ \textit{or}\ \ (t_h \vee t_t) \vee \circ\ \ \textit{or}\ \ t_h \vee t_t \\
\begin{cases} \{t,\ t_h \vee t'_t\} \cap \mathcal{D}om(\Lambda_h \cup \Lambda_t) = \phi & \textit{if}\ \mathcal{S}_{fin_t} = sol \\ \{t,\ t_h \vee t'_t,\ t'_t\} \cap \mathcal{D}om(\Lambda_h \cup \Lambda_t) = \phi & \textit{otherwise} \end{cases}
\end{array}}{\begin{array}{c}
\Big\{\big(t, t_h \vee t'_t, {\mathcal{R}_1}^{\downarrow}, \infty, (\Lambda_h \cup \Lambda'_t)\big)\Big\} \cup (\Lambda_h \cup \Lambda'_t) \vdash t \blacktriangleright {\mathcal{R}_1}^{\downarrow} \\
\textit{where}\ \Lambda_t(t_t) = (t_t, t'_t, {\mathcal{R}_1}^{\downarrow}, \mathcal{S}_{fin_t}, \Lambda'_t)\ \textit{s.t.}\ \mathcal{S}_{fin_t} \in \{sol, \infty\}
\end{array}}\ (\infty)
\end{array}\right\}\ \textit{(T-Dup-impl)}
$$

$$
\left.\begin{array}{c}
\dfrac{\overline{\circ \in \mathcal{E}qu(t) \quad \mathcal{R}_1\ \textit{is Any}}}{\big\{(t, \circ, {\mathcal{R}_1}^{?}, nil, \phi)\big\} \vdash t \blacktriangleright {\mathcal{R}_1}^{?}}\ (nil) \\[3em]
\dfrac{\Lambda' \vdash t' \blacktriangleright \mathcal{R}_1 \quad t' \in \mathcal{E}qu(t)\ \textit{s.t.}\ t \notin \mathcal{D}om(\Lambda')}{\big\{(t, t', {\mathcal{R}_1}^{?}, sol, \Lambda')\big\} \cup \Lambda' \vdash t \blacktriangleright {\mathcal{R}_1}^{?}}\ (sol)
\end{array}\right\}\ \textit{(T-Opt-impl)}
$$

$$
\left.\begin{array}{c}
\dfrac{\Lambda' \vdash t' \blacktriangleright \mathcal{R}_1 \quad \mathcal{R}_2\ \textit{is Any.} \quad t' \in \mathcal{E}qu(t)\ \textit{s.t.}\ t \notin \mathcal{D}om(\Lambda')}{\big\{(t, t', \mathcal{R}_1|\mathcal{R}_2, L, \Lambda')\big\} \cup \Lambda' \vdash t \blacktriangleright \mathcal{R}_1|\mathcal{R}_2}\ (L) \\[3em]
\dfrac{\mathcal{R}_1\ \textit{is Any.} \quad \Lambda'' \vdash t' \blacktriangleright \mathcal{R}_2 \quad t' \in \mathcal{E}qu(t)\ \textit{s.t.}\ t \notin \mathcal{D}om(\Lambda'')}{\big\{(t, t', \mathcal{R}_1|\mathcal{R}_2, R, \Lambda'')\big\} \cup \Lambda'' \vdash t \blacktriangleright \mathcal{R}_1|\mathcal{R}_2}\ (R)
\end{array}\right\}\ \textit{(T-Alt-impl)}
$$

Lemma 5.3 *termination on implementation.*

$$\Lambda \vdash t \blacktriangleright \mathcal{R} \textit{ has termination.}$$

Proof. *Proof is induction on structure of* $(\mathcal{R}, t)$.

From def. of $\Lambda \vdash t \blacktriangleright \mathcal{R}$*, the significant cases to be examined are only (T-Cat0-impl-*∞*), (T-Cat1-impl-*∞*) and (T-Dup-impl-*∞*).*

Case (T-Cat0-impl-∞*): From def. of (T-Cat0-impl-*∞*),* $\Lambda_h \vdash t_h \blacktriangleright \mathcal{R}_1$ *which is one of its subderivations, has termination from induction hypothesis. For the other one, i.e.* $\Lambda_t \vdash t_t \blacktriangleright {\mathcal{R}_1}^*$*, we can say that it also has termination from induction hypothesis according to the reason as below.*

We suppose t_t *s.t.* $\{\} \wedge (t_h \wedge t_t) \in \mathcal{E}qu(t)$*. Then we find* $size(t_t) < size(t)$ *since* $size\big(\{\} \wedge (t_t \wedge t_t)\big) = size\big(\{\}\big) + \big(size(t_h) + size(t_t) + 1\big) + 1 = \big(size(t_h) + size(t_t) + 1\big) + 2 \leq size(t) + 2$ *from Property 5.7, which implies the fact of* $size(t_t) < size(t)$*. Therefore we conclude that* $\Lambda_t \vdash t_t \blacktriangleright {\mathcal{R}_1}^*$ *has termination from induction hypothesis for* $({\mathcal{R}_1}^*, t_t) < ({\mathcal{R}_1}^*, t)$*. The similar case of* $\circ \wedge (t_h \wedge t_t)$ *is also from reasoning as above case.*

For the cases that $t' \in \mathcal{E}qu(t)$ *s.t.* $t = t'' \wedge \{\}$ *or* $t = t'' \wedge \circ$ *for some* t''*, where we let* $t_t = \{\}$ *or* $\circ$ *repectively, from def. of (T-Cat01-impl-*∞*). Consequently* $\Lambda_t \vdash t_t \blacktriangleright {\mathcal{R}_1}^*$ *from (T-Cat0-impl-nil) and (T-Opt-impl-nil), for* $\{\} \in \mathcal{E}qu\big(\{\}\big)$ *and* $\circ \in \mathcal{E}qu(\circ)$ *as respective.*

Finally we cosider other cases, where we let as $t_h \wedge t_t \in \mathcal{E}qu(t)$ *such that both* t_h *and* t_t *are neither* $\{\}$ *nor* $\circ$*. That excludes the cases where the rules, which increases original term size with phony elements of* $\{\}$ *and* $\circ$*, are applied on t of* $\mathcal{E}qu(t)$ *to induce* $t_h \wedge t_t$*. Accordingly* $size(t_t) < size(t)$*, we found from def. of* $\mathcal{E}qu(t)$*. Particularly for the case that the last applied rules are* $\mathcal{A}ssoc_\wedge(t_1, t_2) \subseteq \mathcal{E}qu(t)$ *and* $\mathcal{A}ssoc_\vee(t_1, t_2) \subseteq \mathcal{E}qu(t)$ *where t is* $t_1 \wedge t_2$ *and* $t_1 \vee t_2$ *respectively, see Property 5.7 for the reason of* $size(t') = size(t)$*, where* $t' \in \mathcal{A}ssoc_\wedge(t_1, t_2)$ *and* $t' \in \mathcal{A}ssoc_\vee(t_1, t_2)$ *as respective.*

Case of (T-Cat1-impl-∞) and (T-Dup-impl-∞) are similar to the case of (T-Cat0-impl-∞).

□

Lemma 5.4 *completeness on implementation.*

$$\begin{aligned} &\textit{if } \Lambda \vdash t \blacktriangleright \mathcal{R},\ \textit{then } \Delta \vdash t \triangleright \mathcal{R}\ \textit{s.t. } \mathcal{D}om(\Delta) = \mathcal{D}om(\Lambda), t'_{\Delta} = t'_{\Lambda}, \mathcal{S}_{\Delta} = \mathcal{S}_{\Lambda}, \\ &\qquad\qquad\qquad\qquad \textit{where } \Delta = \big\{(t, t'_{\Delta}, \mathcal{R}, \mathcal{S}_{\Delta}, \Delta')\big\} \cup \Delta' \\ &\qquad\qquad\qquad\qquad\qquad\quad \Lambda = \big\{(t, t'_{\Lambda}, \mathcal{R}, \mathcal{S}_{\Lambda}, \Lambda')\big\} \cup \Lambda' \end{aligned}$$

Proof. *Proof is induction on structure of derivation for $\Lambda \vdash t \blacktriangleright \mathcal{R}$.*

According to the fact that most of rules for implementation have same structure of original ones, we show only significant cases.

Case (T-Cat0-impl-∞): We suppose $\Lambda \vdash t \blacktriangleright {\mathcal{R}_1}^*$ where $\Lambda = \Big\{\big(t, t_h \wedge t'_{t_\Lambda}, {\mathcal{R}_1}^*, \infty, (\Lambda_h \cup \Lambda'_t)\big)\Big\} \cup (\Lambda_h \cup \Lambda'_t)$. From def. of (T-Cat0-impl-∞), we know that its subderivations of $\Lambda_h \vdash t_h \blacktriangleright \mathcal{R}_1$ and $\Lambda_t \vdash t_t \blacktriangleright {\mathcal{R}_1}^*$ implies the existences of $\Delta_h \vdash t_h \triangleright \mathcal{R}_1$ and $\Delta_t \vdash t_t \triangleright {\mathcal{R}_1}^*$ with the conditions of $\mathcal{D}om(\Delta_h) = \mathcal{D}om(\Lambda_h)$, $t'_{h_\Delta} = t'_{h_\Lambda}$, $\mathcal{S}_{h_\Delta} = \mathcal{S}_{h_\Lambda}$ and $\mathcal{D}om(\Delta_t) = \mathcal{D}om(\Lambda_t)$, $t'_{t_\Delta} = t'_{t_\Lambda}$, $\mathcal{S}_{t_\Delta} = \mathcal{S}_{t_\Lambda}$ respectively from induction hypothesis. Consequently, we find $\mathcal{D}om(\Delta_h) \cap \mathcal{D}om(\Delta_t) = \phi$ from the above facts of $\mathcal{D}om(\Delta_h) = \mathcal{D}om(\Lambda_h)$ and $\mathcal{D}om(\Delta_t) = \mathcal{D}om(\Lambda_t)$ accounting on $\mathcal{D}om(\Lambda_h) \cap \mathcal{D}om(\Lambda_t) = \phi$ comming from the def. of (T-Cat0-impl-∞). Next, we see each cases according to the condition on $\mathcal{S}_\Lambda$ as follows. Subcase $\mathcal{S}_{t_\Lambda} \in \{nil, sol\}$: Considering $\{t, t_h \wedge t'_{t_\Lambda}\} \cap \mathcal{D}om(\Lambda_h \cup \Lambda_t) = \phi$ with the facts of $t'_{t_\Delta} = t'_{t_\Lambda}$ and $\mathcal{D}om(\Delta_h \cup \Delta_t) = \mathcal{D}om(\Delta_h) \cup \mathcal{D}om(\Delta_t) = \mathcal{D}om(\Lambda_h) \cup \mathcal{D}om(\Lambda_t) = \mathcal{D}om(\Lambda_h \cup \Lambda_t)$ revealed above, we find $\{t, t_h \wedge t'_{t_\Delta}\} \cap \mathcal{D}om(\Delta_h \cup \Delta_t) = \phi$. Then we know $t \simeq t_h \wedge t_t$ for each case of $t' \in \mathcal{E}qu(t)$ s.t. $t' = \{\} \wedge (t_h \wedge t_t)$ or $t' = \circ \wedge (t_h \wedge t_t)$ or $t' = t_h \wedge t_t$, from Lemma 5.1 with $($RevealH-Cas$_{\{\}})$ or $($RevealH-Cas$_\circ)$. Recalling the fact of $\mathcal{S}_{t_\Delta} = \mathcal{S}_{t_\Lambda}$, we obtain $\Delta \vdash t \triangleright {\mathcal{R}_1}^*$, where we let $\Delta = \Big\{\big(t, t_h \wedge t'_{t_\Delta}, {\mathcal{R}_1}^*, \infty, (\Delta_h \cup \Delta'_t)\big)\Big\} \cup (\Delta_h \cup \Delta'_t)$ s.t. $t_h \wedge t'_{t_\Delta} = t_h \wedge t'_{t_\Lambda}$ and $\mathcal{D}om(\Delta) = \mathcal{D}om(\Lambda)$, since $\mathcal{D}om(\Delta) = \{t\} \cup \mathcal{D}om(\Delta_h \cup \Delta'_t) = \{t\} \cup \big(\mathcal{D}om(\Delta_h) \cup \mathcal{D}om(\Delta'_t)\big) = \{t\} \cup \big(\mathcal{D}om(\Lambda_h) \cup \mathcal{D}om(\Lambda'_t)\big) = \{t\} \cup \mathcal{D}om(\Lambda_h \cup \Lambda'_t) = \mathcal{D}om(\Lambda)$ according to the fact that

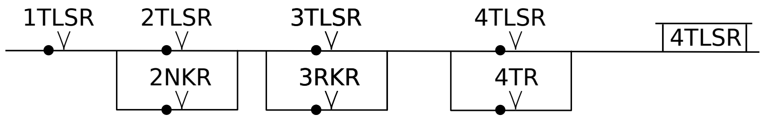
$\{t_t\} \cup \mathcal{Dom}(\Delta'_t) = \{t_t\} \cup \mathcal{Dom}(\Lambda'_t)$ *implies* $\mathcal{Dom}(\Delta'_t) = \mathcal{Dom}(\Lambda'_t)$ *for* $t_t \notin \mathcal{Dom}(\Delta'_t)$ *and* $t_t \notin \mathcal{Dom}(\Lambda'_t)$. *Subcase* $\mathcal{S}_{t_\Lambda} \notin \{nil,\ sol\}$*: It's similar to previous case.* $\square$

6 LIezA, the Logic for Interlocking logics self recognizing Architecture

We implemented our logic on machine using Caml, which is one of the most well-known implementation of ML developped in INRIA France. In this section we introduce it with practical examples.

We can express the following interlocking logic as $TLSR_{1T} \wedge (TLSR_{2T} \vee NKR_{\#2}) \wedge (TLSR_{3T} \vee RKR_{\#3}) \wedge (TLSR_{4T} \vee TR_4)$.

Then, we are now trying to match above with the pattern of $\left(TLSR \vee (NKR|RKR)^?\right)^* \wedge (TLSR \vee TR)$



$$TLSR_{1T} \wedge (TLSR_{2T} \vee NKR_{\#2}) \wedge (TLSR_{3T} \vee RKR_{\#3}) \wedge (TLSR_{4T} \vee TR_4) : \mathcal{R}_{TLSR_{4T}}$$

$$where \;\; \mathcal{R}_{TLSR_{4T}} = \left(TLSR \vee (NKR|RKR)^?\right)^* \wedge (TLSR \vee TR)$$

Figure 4: polymorphic matching of interlocking logic example 1

We can express above instance as `TLSR_1T & (TLSR_2T | NKR_#2) & (TLSR_3T | RKR_#3) & (TLSR_4T | TR_4)`, accroding to def. of term, and pattern is also expressed as `(TLSR | (NKR % RKR)?)* & (TLSR | TR)`. Consequently, we are trying patternmatching, and the result of it is shown as follows.

```
# term (typematch (compile "TLSR_1T & (TLSR_2T | NKR_#2) & (TLS
R_3T | RKR_#3) & (TLSR_4T | TR_4) : (TLSR | (NKR % RKR)?)* & (T
LSR | TR)"));;
- : "(TLSR_1T & ((((TLSR_2T | NKR_#2)) & ((((TLSR_3T | RKR_#3))
 & ((TLSR_4T | TR_4)))))))"
```

Looking at the last line of above, it just been generated from her coginition, which shows that LIezA has successed to match given term with pattern, and recognized its structual meaning, which has been stored as the form of Binding internally. We can still access more specific ones on her cognition as follows.

```
# let sr_trg = cli_pat (compile "* : (TLSR | (NKR % RKR)?)*");;
val sig_trg : Lie_type.pattern = ...
# foreach (resolv sr_trg (typematch (compile "TLSR_1T & (TLSR_2
T | NKR_#2) & (TLSR_3T | RKR_#3) & (TLSR_4T | TR_4) : (TLSR | (
NKR % RKR)?)* & (TLSR | TR)")));;
- : string list = ["(((TLSR_1T | *)) & ((((TLSR_2T | NKR_#2)) &
 ((TLSR_3T | RKR_#3)))))"]
```

The first command of above specifies the target pattern we refer. Then we try to refer the subterms bound to the target pattern, which is currently assigned to `sr_trg`, on next command, And we can find that the last line depicts the subterm bound to the target pattern we specified correctly.

Similar to above, we can access other internal subterms in her cognition, again.

```
# let sr_rel = cli_pat (compile "* : (TLSR | TR)");;
val sig_ctrl : Lie_type.pattern = ...
# foreach (resolv sr_rel (typematch (compile "TLSR_1T & (TLSR_2
T | NKR_#2) & (TLSR_3T | RKR_#3) & (TLSR_4T | TR_4) : (TLSR | (
NKR % RKR)?)* & (TLSR | TR)")));;
- : string list = ["(TLSR_4T | TR_4)"]
```

Finally, we present more practical examples in the domain of railway signaling engineering.

We suppose the following entity of interlocking logic, which is called as Signal Controlling Circuit,

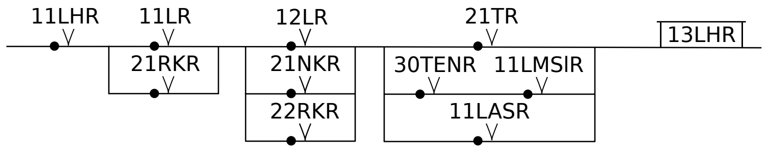


Figure 5: interlocking logic example 2, Signal Control Circult

and we also know which can be expressed generally as $HR \wedge \left(LR \vee (NKR|RKR)^{\downarrow}\right)^{*} \wedge \left(TR^{+} \vee (TENR \wedge MSlR) \vee ASR\right)$.

We express above circuit as `HR_11L&(LR_11|RKR_#21)&(LR_12|NKR_#21|RKR_#22)&(TR_21i|(TENR_30&MSlR_11L)|ASR_11L)`, and try to feed LIezA them as follow.

```
# term (typematch (compile "HR_11L & (LR_11 | RKR_#21) & (LR_12 |
NKR_#21 | RKR_#22) & (TR_21i | (TENR_30 & MSlR_11L) | ASR_11L) :
HR & (LR | (NKR % RKR)!)* & (TR+ | (TENR & MSlR) | ASR)"));;
- : string ="(HR_11L & (((LR_11 | RKR_#21)) & ((((LR_12 |
((NKR_#21 | RKR_#22)))) & ((TR_21i | ((((TENR_30 & MSlR_11L)) |
ASR_11L)))))))))"
```

Then we make confirm her cognition for specific conditions on this circuit. Our pattern for Signal Controlling Circuit has a *ASR* as its subpattern, which corresponds to verification of Approaching Stick Locking a.k.a. ASR on related route's signal aspect controlling. And we can see the condition of ASR verification by searching the one bound to the subpattern of *ASR*, as follow.

```
# let ver_ASR = cli_pat (compile "* : ASR");;
```

```
val ver_ASR : Lie_type.pattern = ...
# foreach (resolv ver_ASR (typematch (compile "HR_11L & (LR_11 |
RKR_21) & (LR_12 | NKR_#21 | RKR_#22) & (TR_21i | (TENR_30 &
MS1R_11L) | ASR_11L) : HR & (LR | (NKR % RKR)!)* & (TR+ | (TENR &
MS1R) | ASR)")));;
- : string list = ["ASR_11L"]
```

LIezA correctly recognized and replied the condition we queried on the last line, as desired.

Similarly we try again to refer the condition of Approaching Track, which is another specific one occuring on Signal Controlling Circuit . And we also obtain correct result from LIezA, as follow.

```
# let ver_appTR = cli_pat (compile "* : TR+");;
val ver_appTR : Lie_type.pattern = ...
# foreach (resolv ver_appTR (typematch (compile "HR_11L & (LR_11 |
RKR_21) & (LR_12 | NKR_#21 | RKR_#22) & (TR_21i | (TENR_30 &
MS1R_11L) | ASR_11L) : HR & (LR | (NKR % RKR)!)* & (TR+ | (TENR &
MS1R) | ASR)")));;
- : string list = ["TR_21i"]
```

7 Appendix 1: supplementary proofs

Definition 7.1

$$\mathcal{G}nd(\Delta) \overset{\text{def}}{:=} \{t_{equ} \mid (t, t_{equ}, \mathcal{R}, \mathcal{S}_{fin}, \Delta') \in \Delta \text{ s.t. } \mathcal{S}_{fin} \in \{gnd, def, nil\}\}$$

Property 7.1

$$gnd\Big(\mathcal{M}\big(\Delta(t)\big)\Big) = \mathcal{G}nd(\Delta) \text{ s.t. } \Delta \vdash t \triangleright \mathcal{R}$$

Proof. *Proof is simple induction on derivation for $\Delta \vdash t \triangleright \mathcal{R}$.* $\square$

Property 7.2

$$\mathcal{D}om(\Delta_1) \cap \mathcal{D}om(\Delta_2) = \phi \text{ where } \Delta_1 \vdash t_1 \triangleright \mathcal{R}_1 \text{ and } \Delta_2 \vdash t_2 \triangleright \mathcal{R}_2, \text{ then}$$
$$sub\Big(\mathcal{M}\big(\Delta_1(t_1)\big)\Big) \cap sub\Big(\mathcal{M}\big(\Delta_2(t_2)\big)\Big) = \phi$$

Proof. *From def. of sub and $\mathcal{M}$, any term in $sub\Big(\mathcal{M}\big(\Delta_1(t_1)\big)\Big)$ consists of the grand terms belonging to $gnd\Big(\mathcal{M}\big(\Delta_1(t_1)\big)\Big)$, i.e. all of grand terms occuring on $t \in sub\Big(\mathcal{M}\big(\Delta_1(t_1)\big)\Big)$ is comming from $gnd\Big(\mathcal{M}\big(\Delta_1(t_1)\big)\Big)$. $sub\Big(\mathcal{M}\big(\Delta_2(t_2)\big)\Big)$ is Similar. Consequently accounting on the facts of $gnd\Big(\mathcal{M}\big(\Delta_1(t_1)\big)\Big) = \mathcal{G}nd(\Delta_1) \subseteq \mathcal{D}om(\Delta_1)$ and $gnd\Big(\mathcal{M}\big(\Delta_1(t_2)\big)\Big) = \mathcal{G}nd(\Delta_2) \subseteq \mathcal{D}om(\Delta_2)$ from Property 7.1 respectively. Thus we find that $gnd\Big(\mathcal{M}\big(\Delta_1(t_1)\big)\Big) \cap gnd\Big(\mathcal{M}\big(\Delta_2(t_2)\big)\Big) = \phi$ for $\mathcal{D}om(\Delta_1) \cap \mathcal{D}om(\Delta_2) = \phi$, which implies $sub\Big(\mathcal{M}\big(\Delta_1(t_1)\big)\Big) \cap sub\Big(\mathcal{M}\big(\Delta_2(t_2)\big)\Big) = \phi$.* $\square$

Lemma 7.1 *completeness on extended matching.*

$$\Gamma \vdash \mathcal{M}\big(\Delta(t)\big) : \mathcal{R} \text{ for some } \Gamma, \text{ where } \Delta \vdash t \triangleright \mathcal{R}.$$

Proof. *Proof is induction on structure of derivation of $\Gamma \vdash t \triangleright \mathcal{R}$.*

Case (T-Atom0-xtend): From def. of (T-Atom0-xtend), we suppose $\{(t, t_0, \mathcal{R}_0, gnd, \phi)\} \vdash t \triangleright \mathcal{R}_0$ s.t. $t \simeq t_0$ with $(t_0, \mathcal{R}_0) \in \Gamma_{gnd}$. Then we can derive $\{(t_0, \mathcal{R}_0)\} \vdash \mathcal{M}\big(\Delta(t)\big) : \mathcal{R}_0$ from (T-Atom0-canon), since $\mathcal{M}\big(\Delta(t)\big) = \mathcal{M}(t, t_0, \mathcal{R}_0, gnd, \phi) = t_0$ from def of $\mathcal{M}$.

Case (T-Atom1-xtend): Proof is same as the case of (T-Atom0-xtend).

Case (T-Cas-xtend): In this case, we suppose that $\Delta \vdash t \rhd \mathcal{R}_1 \wedge \mathcal{R}_2$ *where we let* $\Delta = \Big\{\big(t, t_1 \wedge t_2, \mathcal{R}_1 \wedge \mathcal{R}_2, \wedge, (\Delta_1 \cup \Delta_2)\big)\Big\} \cup (\Delta_1 \cup \Delta_2)$ *for some* Δ_1 *and* Δ_2 *satisfying* $\Delta_1 \vdash t_1 \rhd \mathcal{R}_1$, $\Delta_2 \vdash t_2 \rhd \mathcal{R}_2$ *and* $\mathcal{Dom}(\Delta_1) \cap \mathcal{Dom}(\Delta_2) = \phi$ *from def. of (T-Cas-xtend). By induction hypothesis, we suppose that* $\Gamma_1 \vdash \mathcal{M}\big(\Delta_1(t_1)\big) : \mathcal{R}_1$ *and* $\Gamma_2 \vdash \mathcal{M}\big(\Delta_2(t_2)\big) : \mathcal{R}_2$ *for some* Γ_1 *and* Γ_2 *where* $\Delta_1(t_1) = (t_1, t_{equ_1}, \mathcal{R}_1, \mathcal{S}_1, \Delta_1')$ *and* $\Delta_2(t_2) = (t_2, t_{equ_2}, \mathcal{R}_2, \mathcal{S}_2, \Delta_2')$ *respectively. According to the facts that* $\mathcal{Dom}(\Delta_1 \cup \Delta_2) = \mathcal{Dom}(\Delta_1) \cup \mathcal{Dom}(\Delta_2)$ *s.t.* $\mathcal{Dom}(\Delta_1) \cap \mathcal{Dom}(\Delta_2) = \phi$*, we find* $t_1 \notin \mathcal{Dom}(\Delta_2)$ *s.t.* $t_1 \in \mathcal{Dom}(\Delta_1)$ *for* $\Delta_1 \vdash t_1 \rhd \mathcal{R}_1$*, and* $t_2 \notin \mathcal{Dom}(\Delta_1)$ *s.t* $t_2 \in \mathcal{Dom}(\Delta_2)$ *for* $\Delta_2 \vdash t_2 \rhd \mathcal{R}_2$*. Thus* $\mathcal{M}\big(\Delta(t)\big) = \mathcal{M}\big(t, t_1 \wedge t_2, \mathcal{R}_1 \wedge \mathcal{R}_2, \wedge, (\Delta_1 \cup \Delta_2)\big) = \mathcal{M}\big((\Delta_1 \cup \Delta_2)(t_1)\big) \wedge \mathcal{M}\big((\Delta_1 \cup \Delta_2)(t_2)\big) = \mathcal{M}\big(\Delta_1(t_1)\big) \wedge \mathcal{M}\big(\Delta_2(t_2)\big)$ *allows us to conclude* $\Big\{\Big(\mathcal{M}\big(\Delta_1(t_1)\big) \wedge \mathcal{M}\big(\Delta_2(t_2)\big), \mathcal{R}_1 \wedge \mathcal{R}_2\Big)\Big\} \cup (\Gamma_1 \cup \Gamma_2) \vdash \mathcal{M}\big(\Delta_1(t_1)\big) \wedge \mathcal{M}\big(\Delta_2(t_2)\big) : \mathcal{R}_1 \wedge \mathcal{R}_2$ *from (T-Cas-canon), with the conditions of* $\mathcal{Dom}(\Gamma_1) \cap \mathcal{Dom}(\Gamma_2) = \phi$ *and* $\mathcal{M}\big(\Delta_1(t_1)\big) \wedge \mathcal{M}\big(\Delta_2(t_2)\big) \notin \mathcal{Dom}(\Gamma_1 \cup \Gamma_2)$ *as follows. We find that* $sub\Big(\mathcal{M}\big(\Delta_1(t_1)\big)\Big) \cap sub\Big(\mathcal{M}\big(\Delta_2(t_2)\big)\Big) = \phi$ *from the facts of* $\mathcal{Dom}(\Delta_1) \cap \mathcal{Dom}(\Delta_2) = \phi$ *s.t.* $\Delta_1 \vdash t_1 \rhd \mathcal{R}_1$ *and* $\Delta_2 \vdash t_2 \rhd \mathcal{R}_2$ *with Property 7.2, then we even do* $\mathcal{Dom}(\Gamma_1) \subseteq sub\Big(\mathcal{M}\big(\Delta_1(t_1)\big)\Big)$ *and* $\mathcal{Dom}(\Gamma_2) \subseteq sub\Big(\mathcal{M}\big(\Delta_2(t_2)\big)\Big)$ *where* $\Gamma_1 \vdash \mathcal{M}\big(\Delta_1(t_1)\big) : \mathcal{R}_1$ *and* $\Gamma_2 \vdash \mathcal{M}\big(\Delta_2(t_2)\big) : \mathcal{R}_2$ *respectively from Property 3.4, thus* $\mathcal{Dom}(\Gamma_1) \cap \mathcal{Dom}(\Gamma_2) = \phi$*. Consequently, we show the reason of* $\mathcal{M}\big(\Delta_1(t_1)\big) \wedge \mathcal{M}\big(\Delta_2(t_2)\big) \notin \mathcal{Dom}(\Gamma_1 \cup \Gamma_2)$*. We know that* $size\Big(\mathcal{M}\big(\Delta_1(t_1)\big) \wedge \mathcal{M}\big(\Delta_2(t_2)\big)\Big) > size\Big(\mathcal{M}\big(\Delta_1(t_1)\big)\Big) \geq size(t_1')$ *for any* $t_1' \in \mathcal{Dom}(\Gamma_1)$ *from Property 3.5, which implies that* $\mathcal{M}\big(\Delta_1(t_1)\big) \wedge \mathcal{M}\big(\Delta_2(t_2)\big) \notin \mathcal{Dom}(\Gamma_1)$ *and* $\mathcal{M}\big(\Delta_1(t_1)\big) \wedge \mathcal{M}\big(\Delta_2(t_2)\big) \notin \mathcal{Dom}(\Gamma_2)$ *respectively, thus we obtain* $\mathcal{M}\big(\Delta_1(t_1)\big) \wedge \mathcal{M}\big(\Delta_2(t_2)\big) \notin \Big(\mathcal{Dom}(\Gamma_1) \cup \mathcal{Dom}(\Gamma_2)\Big) = \mathcal{Dom}(\Gamma_1 \cup \Gamma_2)$*. Note that we call the reason of above, i.e.* $\mathcal{M}\big(\Delta_1(t_1)\big) \wedge \mathcal{M}\big(\Delta_2(t_2)\big) \notin \mathcal{Dom}(\Gamma_1 \cup \Gamma_2)$*, as "term size constraint" in below argument.*

Case (T-Par-xtend): Proof is same as the case of (T-Cas-xtend).

Case (T-Cat0-xtend-nil): Similar to the case of (T-Atom0-xtend) as follow. We suppose $\Big\{\big(t, \{\}, \mathcal{R}_1{}^*, nil, \phi\big)\Big\} \vdash t \rhd \mathcal{R}_1{}^*$ *s.t.* $t \simeq \{\}$*. Thus we can derive* $\Big\{\big(\{\}, \mathcal{R}_1{}^*\big)\Big\} \vdash \mathcal{M}\big(\Delta(t)\big) : \mathcal{R}_0{}^*$

from (T-Cat0-canon-nil), since $\mathcal{M}\big(\Delta(t)\big) = \mathcal{M}\big(t, \{\}, \mathcal{R}_0{}^*, nil, \phi\big) = \{\}$ *from def of* $\mathcal{M}$*.*

Case (T-Cat0-xtend-sol): In this case, we suppose the deduction of $\Delta \vdash t \rhd \mathcal{R}_1{}^*$ *from (T-Cat0-xtend-sol) where we let* $\Delta = \big\{(t, t', \mathcal{R}_1{}^*, sol, \Delta')\big\} \cup \Delta'$*, which also implies the existence of derivation of* $\Delta' \vdash t' \rhd \mathcal{R}_1$ *s.t.* $t \simeq t'$ *and* $t \notin \mathcal{Dom}(\Delta')$*. for some* Δ'*. According to the fact that* $\Gamma' \vdash \mathcal{M}\big(\Delta'(t')\big) : \mathcal{R}_1$ *for some* Γ' *obtained by applying induction hypothesis on* $\Delta' \vdash t' \rhd \mathcal{R}_1$*, we have* $\left\{\left(\left\{\mathcal{M}\big(\Delta'(t')\big)\right\}, \mathcal{R}_1{}^*\right)\right\} \cup \Gamma' \vdash \left\{\mathcal{M}\big(\Delta'(t')\big)\right\} : \mathcal{R}_1{}^*$ *from (T-Cat0-canon-*∞*), since* $\mathcal{M}\big(\Delta(t)\big) = \mathcal{M}(t, t', \mathcal{R}_1{}^*, sol, \Delta') = \left\{\mathcal{M}\big(\Delta'(t')\big)\right\}$ *s.t* $\left\{\mathcal{M}\big(\Delta'(t')\big)\right\} \notin \mathcal{Dom}(\Gamma')$ *by term size constraint.*

Case (T-Cat0-xtend-∞*): In this case, we suppose the last applied on deduction of* $\Delta \vdash t \rhd \mathcal{R}_1{}^*$ *is (T-Cat0-xtend-*∞*), where we let* $\Delta = \Big\{\big(t, t_h \wedge t'_t, \mathcal{R}_1{}^*, \infty, (\Delta_h \cup \Delta'_t)\big)\Big\} \cup (\Delta_h \cup \Delta'_t)$ *s.t.* $\Delta_h = \big\{(t_h, t'_h, \mathcal{R}_1, \mathcal{S}_h, \Delta'_h)\big\} \cup \Delta'_h$ *and* $\Delta_t = \big\{(t_t, t'_t, \mathcal{R}_1{}^*, \mathcal{S}_t, \Delta'_t)\big\} \cup \Delta'_t$ *comming from* $\Delta_h \vdash t_h \rhd \mathcal{R}_1$ *and* $\Delta_t \vdash t_t \rhd \mathcal{R}_1{}^*$ *as respective. Additionally, there are 2 cases to be considered, focusing on the derivation of* $\Delta_t \vdash t_t \rhd \mathcal{R}_1{}^*$ *as follows. Subcase (T-Cat0-xtend-sol): In this case, we suppose that the last applied rule on the derivation of* $\Delta_t \vdash t_t \rhd \mathcal{R}_1{}^*$ *is (T-Cat0-xtend-sol), where* $\Delta_t = \big\{(t_t, t'_t, \mathcal{R}_1{}^*, sol, \Delta'_t)\big\} \cup \Delta'_t$ *and* $\Delta'_t = (t'_t, t''_t, \mathcal{R}_1, \mathcal{S}_{t'}, \Delta''_t)$ *s.t.* $\Delta'_t \vdash t'_t \rhd \mathcal{R}_1$ *we let. By applying induction hypothesis on* $\Delta_h \vdash t_h \rhd \mathcal{R}_1$*, we find* $\Gamma_h \vdash \mathcal{M}\big(\Delta_h(t_h)\big) : \mathcal{R}_1$ *for some* Γ_h*. Similarly, we can also obtain* $\Gamma'_t \vdash \mathcal{M}\big(\Delta'_t(t'_t)\big) : \mathcal{R}_1$ *for some* Γ'_t *from induction hypothesis for* $\Delta'_t \vdash t'_t \rhd \mathcal{R}_1$*. Accounting on the fact that* $\mathcal{Dom}(\Delta'_t) \subseteq \mathcal{Dom}(\Delta_t)$ *where* $\Delta_t = \big\{(t_t, t'_t, \mathcal{R}_1{}^*, sol, \Delta'_t)\big\} \cup \Delta'_t$*, we find* $\mathcal{Dom}(\Delta_h) \cap \mathcal{Dom}(\Delta'_t) = \phi$*, because of* $\mathcal{Dom}(\Delta_h) \cap \mathcal{Dom}(\Delta_t) = \phi$ *from the def. of (T-Cat0-xtend-*∞*). Then we can say that* $sub\Big(\mathcal{M}\big(\Delta_h(t_h)\big)\Big) \cap sub\Big(\mathcal{M}\big(\Delta'_t(t'_t)\big)\Big) = \phi$ *from Lemma 7.2 for* $\Delta_h \vdash t_h \rhd \mathcal{R}_1$ *and* $\Delta'_t \vdash t'_t \rhd \mathcal{R}_1$ *above. By now, recalling the facts we revealed above as* $\Gamma_h \vdash \mathcal{M}\big(\Delta_h(t_h)\big) : \mathcal{R}_1$ *and* $\Gamma'_t \vdash \mathcal{M}\big(\Delta'_t(t'_t)\big) : \mathcal{R}_1$ *for some* Γ_h *and* Γ'_t *respectively, we still find that* $\mathcal{Dom}(\Gamma_h) \subseteq sub\Big(\mathcal{M}\big(\Delta_h(t_h)\big)\Big)$ *and* $\mathcal{Dom}(\Gamma'_t) \subseteq sub\Big(\mathcal{M}\big(\Delta'_t(t'_t)\big)\Big)$ *from Property 3.4, thus we find* $\mathcal{Dom}(\Gamma_h) \cap \mathcal{Dom}(\Gamma'_t) = \phi$*. Accordingly we obtain the conclusion* *that* $\Big\{\Big(\{\mathcal{M}\big(\Delta_h(t_h)\big), \mathcal{M}\big(\Delta'_t(t'_t)\big)\}, \mathcal{R}_1{}^*\Big)\Big\} \cup (\Gamma_h \cup \Gamma'_t) \vdash \Big\{\mathcal{M}\big(\Delta_h(t_h)\big), \mathcal{M}\big(\Delta'_t(t'_t)\big)\Big\} : \mathcal{R}_1{}^*$ *from (T-Cat0-canon-∞), where we mention that* $\mathcal{M}\big(\Delta(t)\big) = \mathcal{M}\big(t, t_h \wedge t'_t, \mathcal{R}_1{}^*, \infty, (\Delta_h \cup \Delta'_t)\big) = \Big\{\mathcal{M}\big((\Delta_h \cup \Delta'_t)(t_h)\big), \mathcal{M}\big((\Delta_h \cup \Delta'_t)(t'_t)\big)\Big\} = \Big\{\mathcal{M}\big(\Delta_h(t_h)\big), \mathcal{M}\big(\Delta'_t(t'_t)\big)\Big\}$ *since* $(\Delta_h \cup \Delta'_t)(t_h) = \Delta_h(t_h)$ *and* $(\Delta_h \cup \Delta'_t)(t'_t) = \Delta'_t(t'_t)$ *from the fact of* $\mathcal{D}om(\Delta_h) \cap \mathcal{D}om(\Delta'_t) = \phi$ *as above, thus* $\Big\{\mathcal{M}\big(\Delta_h(t_h)\big), \mathcal{M}\big(\Delta'_t(t'_t)\big)\Big\} \notin \mathcal{D}om(\Gamma_h)$ *and* $\Big\{\mathcal{M}\big(\Delta_h(t_h)\big), \mathcal{M}\big(\Delta'_t(t'_t)\big)\Big\} \notin \mathcal{D}om(\Gamma'_t)$ *s.t.* $\Gamma_h \vdash \mathcal{M}\big(\Delta_h(t_h)\big) : \mathcal{R}_1$ *and* $\Gamma'_t \vdash \mathcal{M}\big(\Delta'_t(t'_t)\big) : \mathcal{R}_1$ *respectively by term size constraint. i.e.* $size\bigg(\Big\{\mathcal{M}\big(\Delta_h(t_h)\big), \mathcal{M}\big(\Delta'_t(t'_t)\big)\Big\}\bigg) > size\Big(\mathcal{M}\big(\Delta_h(t_h)\big)\Big) \geq size(t')$ *for any* $t' \in \mathcal{D}om(\Gamma_h)$ *s.t.* $\Gamma_h \vdash \mathcal{M}\big(\Delta_h(t_h)\big) : \mathcal{R}_1$*, which implies* $\Big\{\mathcal{M}\big(\Delta_h(t_h)\big), \mathcal{M}\big(\Delta'_t(t'_t)\big)\Big\} \notin \mathcal{D}om(\Gamma_h)$*,* $\Big\{\mathcal{M}\big(\Delta_h(t_h)\big), \mathcal{M}\big(\Delta'_t(t'_t)\big)\Big\} \notin \mathcal{D}om(\Gamma'_t)$ *is similar. Therefore* $\{\mathcal{M}\big(\Delta_h(t_h)\big), \mathcal{M}\big(\Delta'_t(t'_t)\big)\} \notin \big(\mathcal{D}om(\Gamma_h) \cup \mathcal{D}om(\Gamma'_t)\big)$ *as desired. Subcase (T-Cat0-xtend-∞): Otherwise, i.e. in the case of that the last applied rule on* $\Delta_t \vdash t_t \rhd \mathcal{R}_1{}^*$ *isn't (T-Cat0-xtend-sol), we can predicate that the last one applied on* $\Delta_t \vdash t_t \rhd \mathcal{R}_1{}^*$ *is just (T-Cat0-xtend-∞), since the constraint of* $t'_t \neq \{\}$ *on* $\Delta_t(t_t) = (t_t, t'_t, \mathcal{R}_1{}^*, \mathcal{S}_t, \Delta'_t)$ *eliminates the case of (T-Cat0-xtend-nil) as the last applied rule. Accordingly we suppose* $\Delta_t = \big\{(t_t, t'_t, \mathcal{R}_1{}^*, \infty, \Delta'_t)\big\} \cup \Delta'_t$ *s.t.* $t'_t = t_{t_h} \wedge t'_{t_t}$ *and* $\Delta'_t = \Delta_{t_h} \cup \Delta'_{t_t}$*, where we let* $\Delta_{t_h} = \big\{(t_{t_h}, t'_{t_h}, \mathcal{R}_1, \mathcal{S}_{t_h}, \Delta'_{t_h})\big\} \cup \Delta'_{t_h}$*, and* $\Delta_{t_t} = \big\{(t_{t_t}, t'_{t_t}, \mathcal{R}_1{}^*, \mathcal{S}_{t_t}, \Delta'_{t_t})\big\} \cup \Delta'_{t_t}$ *s.t.* $\Delta_{t_h} \vdash t_{t_h} \rhd \mathcal{R}_1$ *and* $\Delta_{t_t} \vdash t_{t_t} \rhd \mathcal{R}_1{}^*$ *respectively comming from subderivations of* $\Delta_t \vdash t_t \rhd \mathcal{R}_1{}^*$ *by (T-Cat0-xtend-∞). Based on above assumptions with the fact* $t'_t \notin \mathcal{D}om(\Delta_h \cup \Delta'_t)$ *since* $\mathcal{D}om(\Delta_h \cup \Delta'_t) = \big(\mathcal{D}om(\Delta_h) \cup \mathcal{D}om(\Delta'_t)\big) \subseteq \big(\mathcal{D}om(\Delta_h) \cup \mathcal{D}om(\Delta_t)\big) = \mathcal{D}om(\Delta_h \cup \Delta_t)$ *s.t.* $t'_t \notin \mathcal{D}om(\Delta_h \cup \Delta_t)$ *from def. of (T-Cat0-xtend-∞), we get* $\mathcal{M}\big(\Delta(t)\big) = \mathcal{M}(t, t_h \wedge t'_t, \mathcal{R}_1{}^*, \infty, \Delta_h \cup \Delta'_t) = \Big\{\mathcal{M}\big((\Delta_h \cup \Delta'_t)(t_h)\big), t_2, \cdots t_n\Big\} = \Big\{\mathcal{M}\big(\Delta_h(t_h)\big), t_2, \cdots t_n\Big\}$ *satisfying* $\{t_2, \cdots t_n\} = \mathcal{M}_{\{\}}\bigg(t'_t, \mathcal{R}_1{}^*, \Big((\Delta_h \cup \Delta'_t) \setminus \big((\Delta_h \cup \Delta'_t)(t_h) \cup \Delta'_h\big)\Big)\bigg) = \mathcal{M}_{\{\}}\bigg(t'_t, \mathcal{R}_1{}^*, \Big((\Delta_h \cup \Delta'_t) \setminus \big(\Delta_h(t_h) \cup \Delta'_h\big)\Big)\bigg) = \mathcal{M}_{\{\}}\Big(t'_t, \mathcal{R}_1{}^*, \big((\Delta_h \cup \Delta'_t) \setminus \Delta_h\big)\Big) = \mathcal{M}_{\{\}}(t'_t, \mathcal{R}_1{}^*, \Delta'_t)$ *where* $(\Delta_h \cup \Delta'_t)(t_h) = \Delta_h(t_h) = (t_h, t'_h, \mathcal{R}_1, \mathcal{S}_h, \Delta'_h)$ *for* $t_h \notin \mathcal{D}om(\Delta'_t)$*. Note that* $\Delta_h \vdash t_h \rhd \mathcal{R}_1$ *implies* $t_h \in \mathcal{D}om(\Delta_h)$*, thus we can say* $t_h \notin \mathcal{D}om(\Delta'_t)$

for $\mathcal{Dom}(\Delta'_t) \subseteq \mathcal{Dom}(\Delta_t)$ s.t. $\mathcal{Dom}(\Delta_h) \cap \mathcal{Dom}(\Delta_t) = \phi$. From def. of (T-Cat0-xtend-∞) applied on $\Delta_t \vdash t_t \triangleright \mathcal{R}_1{}^*$, we know that $t'_t = t_{t_h} \wedge t'_{t_t} \notin \big(\mathcal{Dom}(\Delta_{t_h}) \cup \mathcal{Dom}(\Delta'_{t_t})\big) = \mathcal{Dom}(\Delta'_t)$ since $\big(\mathcal{Dom}(\Delta_{t_h}) \cup \mathcal{Dom}(\Delta'_{t_t})\big) \subseteq \big(\mathcal{Dom}(\Delta_{t_h}) \cup \mathcal{Dom}(\Delta_{t_t})\big) = \mathcal{Dom}(\Delta_{t_h} \cup \Delta_{t_t})$, then we look for $\{t_2, \cdots t_n\} = \mathcal{M}_{\{\}}(t'_t, \mathcal{R}_1{}^*, \Delta'_t) = \Big\{\mathcal{M}\big(\Delta'_t(t_{t_h})\big), t'_2, \cdots t'_m\Big\} = \Big\{\mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big), t'_2, \cdots t'_m\Big\}$ where $\{t'_2, \cdots t'_m\} = \mathcal{M}_{\{\}}\Big(t'_{t_t}, \mathcal{R}_1{}^*, \Big(\Delta'_t \backslash \big(\Delta'_t(t_{t_h}) \cup \Delta'_{t_h}\big)\Big)\Big)$ s.t. $\Delta'_t(t_{t_h}) = (\Delta_{t_h} \cup \Delta'_{t_t})(t_{t_h}) = \Delta_{t_h}(t_{t_h}) = (t_{t_h}, t'_{t_h}, \mathcal{R}_1, \mathcal{S}_{t_h}, \Delta'_{t_h})$ since $t_{t_h} \notin \mathcal{Dom}(\Delta'_{t_t})$ for the same reason of $t_h \notin \mathcal{Dom}(\Delta'_t)$ as above. Then we can apply Claim 7.1 on $\{t'_2, \cdots t'_m\} = \mathcal{M}_{\{\}}\Big(t'_{t_t}, \mathcal{R}_1{}^*, \Big(\Delta'_t \backslash \big(\Delta'_t(t_{t_h}) \cup \Delta'_{t_h}\big)\Big)\Big)$, we obtain the facts that $\Gamma'_2 \vdash t'_2 : \mathcal{R}_1, \cdots \Gamma'_m \vdash t'_m : \mathcal{R}_1$ for some $\Gamma'_2, \cdots \Gamma'_m$ s.t. $\mathcal{Dom}(\Gamma'_j) \cap \mathcal{Dom}(\Gamma'_k) = \phi$ where $j \neq k$, and $\bigcup_{j=2}^m \mathcal{Dom}(\Gamma'_j) \subseteq \mathit{sub}\Big(\mathcal{M}\big(\Delta_{t_t}(t_{t_t})\big)\Big)$. According to the facts of $\mathcal{Dom}(\Delta_{t_h}) \cap \mathcal{Dom}(\Delta_{t_t}) = \phi$ s.t. $\Delta_{t_h} \vdash t_{t_h} \triangleright \mathcal{R}_1$, $\Delta_{t_t} \vdash t_{t_t} \triangleright \mathcal{R}_1{}^*$, we get $\mathit{sub}\Big(\mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big)\Big) \cap \mathit{sub}\Big(\mathcal{M}\big(\Delta_{t_t}(t_{t_t})\big)\Big) = \phi$ with $\mathcal{Dom}(\Gamma_{t_h}) \subseteq \mathit{sub}\Big(\mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big)\Big)$ s.t. $\Gamma_{t_h} \vdash \mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big) : \mathcal{R}_1$ from Property 7.2, 3.4 and induction hypothesis applied on $\Delta_{t_h} \vdash t_{t_h} \triangleright \mathcal{R}_1$ respectively. Thus we still find $\mathcal{Dom}(\Gamma_{t_h}) \cap \bigcup_{j=2}^m \mathcal{Dom}(\Gamma'_j) = \phi$ for $\bigcup_{j=2}^m \mathcal{Dom}(\Gamma'_j) \subseteq \mathit{sub}\Big(\mathcal{M}\big(\Delta_{t_t}(t_{t_t})\big)\Big)$ above. Accounting on the facts that $\mathcal{Dom}(\Delta_{t_h}) \subseteq \mathcal{Dom}(\Delta_t)$ s.t. $\mathcal{Dom}(\Delta_h) \cap \mathcal{Dom}(\Delta_t) = \phi$, which implies $\mathcal{Dom}(\Delta_h) \cap \mathcal{Dom}(\Delta_{t_h}) = \phi$, thus we find that $\mathit{sub}\Big(\mathcal{M}\big(\Delta_h(t_h)\big)\Big) \cap \mathit{sub}\Big(\mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big)\Big) = \phi$ similarly to above argument with Property 7.2. Since $\mathcal{Dom}(\Gamma_h) \subseteq \mathit{sub}\Big(\mathcal{M}\big(\Delta_h(t_h)\big)\Big)$ s.t. $\Gamma_h \vdash \mathcal{M}\big(\Delta_h(t_h)\big) : \mathcal{R}_1$ and $\mathcal{Dom}(\Gamma_{t_h}) \subseteq \mathit{sub}\Big(\mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big)\Big)$ s.t. $\Gamma_{t_h} \vdash \mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big) : \mathcal{R}_1$ both are from Property 3.4, then we obtain $\mathcal{Dom}(\Gamma_{t_h}) \cap \big(\mathcal{Dom}(\Gamma_h) \cup \bigcup_{j=2}^m \mathcal{Dom}(\Gamma'_j)\big) = \phi$ for $\mathcal{Dom}(\Gamma_{t_h}) \cap \bigcup_{j=2}^m \mathcal{Dom}(\Gamma'_j) = \phi$ above. Similarly according to the facts of $\mathcal{Dom}(\Delta_h) \cap \mathcal{Dom}(\Delta_t) = \phi$ s.t. $\Delta_h \vdash t_h \triangleright \mathcal{R}_1$ and $\Delta_t \vdash t_t \triangleright \mathcal{R}_1{}^*$, we see that $\mathcal{Dom}(\Gamma_h) \cap \mathcal{Dom}(\Gamma_t) = \phi$ s.t $\Gamma_h \vdash \mathcal{M}\big(\Delta_h(t_h)\big) : \mathcal{R}_1$ and $\Gamma_t \vdash \mathcal{M}\big(\Delta_t(t_t)\big) : \mathcal{R}_1{}^*$ since $\mathit{sub}\Big(\mathcal{M}\big(\Delta_h(t_h)\big)\Big) \cap \mathit{sub}\Big(\mathcal{M}\big(\Delta_t(t_t)\big)\Big) = \phi$ s.t. $\mathcal{Dom}(\Gamma_h) \subseteq \mathit{sub}\Big(\mathcal{M}\big(\Delta_h(t_h)\big)\Big)$ and $\mathcal{Dom}(\Gamma_t) \subseteq \mathit{sub}\Big(\mathcal{M}\big(\Delta_t(t_t)\big)\Big)$ from Property 7.2 and 3.4 respectively. Next we focus on $\mathcal{M}\big(\Delta_t(t_t)\big)$ to show $\bigcup_{j=2}^m \mathcal{Dom}(\Gamma'_j) \subseteq \mathit{sub}\Big(\mathcal{M}\big(\Delta_t(t_t)\big)\Big)$. From def of $\mathcal{M}$, we know that $\mathcal{M}\big(\Delta_t(t_t)\big) = \mathcal{M}(t_t, t'_t, \mathcal{R}_1{}^*, \infty, \Delta'_t) = \mathcal{M}(t_t, t_{t_h} \wedge t'_{t_t}, \mathcal{R}_1{}^*, \infty, \Delta'_t) = \Big\{\mathcal{M}\big(\Delta'_t(t_{t_h})\big), t''_2, \cdots t''_l\Big\} = \Big\{\mathcal{M}\big((\Delta_{t_h} \cup \Delta'_{t_t})(t_{t_h})\big), t''_2, \cdots t''_l\Big\} = \Big\{\mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big), t''_2, \cdots t''_l\Big\}$

where $\{t''_2, \cdots t''_l\} = \mathcal{M}_{\{\}}\Big(t'_{t_t}, {\mathcal{R}_1}^*, \big(\Delta'_t \setminus \big(\Delta'_t(t_{t_h}) \cup \Delta'_{t_h}\big)\big)\Big) = \{t'_2, \cdots t'_m\}$ *shown as above. Thus* $\mathit{sub}\Big(\mathcal{M}\big(\Delta_t(t_t)\big)\Big) = \mathit{sub}\Big(\Big\{\mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big), t'_2, \cdots t'_m\Big\}\Big) \supseteq \bigcup_{j=2}^m \mathit{sub}(t'_j) \supseteq \bigcup_{j=2}^m \mathit{Dom}(\Gamma'_j)$ *thus* $\mathit{sub}\Big(\mathcal{M}\big(\Delta_t(t_t)\big)\Big) \supseteq \bigcup_{j=2}^m \mathit{Dom}(\Gamma'_j)$ *s.t.* $\Gamma'_j \vdash t'_j : \mathcal{R}_1$ *for each* $2 \leq j \leq m$*. Therefore* $\mathit{Dom}(\Gamma_h) \cap \bigcup_{j=2}^m \mathit{Dom}(\Gamma'_j) = \phi$ *from* $\mathit{sub}\Big(\mathcal{M}\big(\Delta_h(t_h)\big)\Big) \cap \mathit{sub}\Big(\mathcal{M}\big(\Delta_t(t_t)\big)\Big) = \phi$ *with* $\mathit{Dom}(\Gamma_h) \subseteq \mathit{sub}\Big(\mathcal{M}\big(\Delta_h(t_h)\big)\Big)$ *revealed above, which even implies that* $\mathit{Dom}(\Gamma_h) \cap \big(\mathit{Dom}(\Gamma_{t_h}) \cup \bigcup_{j=2}^m \mathit{Dom}(\Gamma'_j)\big) = \phi$ *since* $\mathit{Dom}(\Gamma_h) \cap \mathit{Dom}(\Gamma_{t_h}) = \phi$ *as above. Consequently we have revealed that* $\mathit{Dom}(\Gamma'_j) \cap \mathit{Dom}(\Gamma'_k) = \phi$ *s.t.* $j \neq k$, $\mathit{Dom}(\Gamma_{t_h}) \cap \bigcup_{j=2}^m \mathit{Dom}(\Gamma'_j) = \phi$ *and* $\mathit{Dom}(\Gamma_h) \cap \big(\mathit{Dom}(\Gamma_{t_h}) \cup \bigcup_{j=2}^m \mathit{Dom}(\Gamma'_j)\big) = \phi$*, which allow us to conclude as* $\Big\{\Big(\mathcal{M}\big(\Delta(t)\big), {\mathcal{R}_1}^*\Big)\Big\} \cup \big(\Gamma_h \cup \Gamma_{t_h} \cup \bigcup_{j=2}^m \Gamma'_j\big) \vdash \mathcal{M}\big(\Delta(t)\big) : {\mathcal{R}_1}^*$ *where* $\mathcal{M}\big(\Delta(t)\big) = \Big\{\mathcal{M}\big(\Delta_h(t_h)\big), t_2, \cdots t_n\Big\} = \Big\{\mathcal{M}\big(\Delta_h(t_h)\big), \mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big), t'_2, \cdots t'_m\Big\}$ *s.t.* $\Gamma_h \vdash \mathcal{M}\big(\Delta_h(t_h)\big) : \mathcal{R}_1$*, for some* Γ_h*,* $\Gamma_{t_h} \vdash \mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big) : \mathcal{R}_1$*, for some* Γ_{t_h}*, and* $\Gamma'_j \vdash t'_j : \mathcal{R}_1$ *for each* $2 \leq j \leq m$ *respectively. We mention that* $\mathcal{M}\big(\Delta(t)\big) = \Big\{\mathcal{M}\big(\Delta_h(t_h)\big), \mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big), t'_2, \cdots t'_m\Big\} \notin \mathit{Dom}\big(\Gamma_h \cup \Gamma_{t_h} \cup \bigcup_{j=2}^m \Gamma'_j\big)$ *by size constraint, i.e.* $\mathit{size}\Big(\mathcal{M}\big(\Delta(t)\big)\Big) > \mathit{size}\Big(\mathcal{M}\big(\Delta_h(t_h)\big)\Big)$, $\mathit{size}\Big(\mathcal{M}\big(\Delta_{t_h}(t_{t_h})\big)\Big)$, $\mathit{size}(t'_2), \cdots \mathit{size}(t'_m)$*, which implies* $\mathcal{M}\big(\Delta(t)\big) \notin \mathit{Dom}\big(\Gamma_h \cup \Gamma_{t_h} \cup \bigcup_{j=2}^m \Gamma'_j\big)$.

Case (T-Cat1-xtend-sol): Proof is similar to the case of (T-Cat0-xtend-sol).

Case (T-Cat1-xtend-∞*): Proof is similar to the case of (T-Cat0-xtend-*∞*).*

Case (T-Dup-xtend-sol): Proof is similar to the case of case (T-Cat1-xtend-sol).

Case (T-Dup-xtend-∞*): Proof is similar to the case of (T-Cat1-xtend-*∞*).*

Case (T-Opt-xtend-nil): Proof is similar to the case of (T-Cat0-xtend-nil), from the def of $\mathcal{M}$ *as* $\mathcal{M}\big(\Delta(t)\big) = \mathcal{M}(t, \circ, {\mathcal{R}_1}^?, \mathit{nil}, \phi) = \circ$*, with (T-Opt-canon-nil).*

Case (T-Opt-xtend-sol): We suppose $\Delta \vdash t \triangleright {\mathcal{R}_1}^?$ *where we let* $\Delta = \big\{(t, t', {\mathcal{R}_1}^?, \mathit{sol}, \Delta')\big\} \cup \Delta'$ *s.t* $\Delta' \vdash t' \triangleright \mathcal{R}_1$ *from def. of (T-Opt-xtend-sol). Then from def. of* $\mathcal{M}$*, we obtain* $\mathcal{M}\big(\Delta(t)\big) = \mathcal{M}(t, t', {\mathcal{R}_1}^?, \mathit{sol}, \Delta') = \mathcal{M}\big(\Delta'(t')\big)^?$ *s.t.* $\Gamma' \vdash \mathcal{M}\big(\Delta'(t')\big) : \mathcal{R}_1$ *for some* Γ' *by induction hypothesis appled on* $\Delta' \vdash t' \triangleright \mathcal{R}_1$*, which also satisfies the condition that* $\mathcal{M}\big(\Delta'(t')\big)^? \notin \mathit{Dom}(\Gamma')$

by term size constraint. Thus we obtain $\Big\{\Big(\mathcal{M}\big(\Delta'(t')\big)^?, \mathcal{R}_1{}^?\Big)\Big\} \cup \Gamma' \vdash \mathcal{M}\big(\Delta'(t')\big)^? : \mathcal{R}_1{}^?$ *from (T-Opt-canon-sol).*

Case (T-Alt-xtend-L): Proof is similar to the case of (T-Opt-xtend-sol), with (T-Alt-canon-L).

Case (T-Alt-xtend-R): Proof is similar to the case of (T-Alt-xtend-L), with (T-Alt-canon-R).

□

Claim 7.1

$$\{t_2, \cdots t_n\} = \mathcal{M}_{\{\}}\Big(t'_{equ_t}, \mathcal{R}_1{}^*, \big(\Delta' \setminus \big(\Delta'(t_{equ_h}) \cup \Delta'_h\big)\big)\Big)$$

where $\Delta \vdash t \triangleright \mathcal{R}_1{}^*$ *s.t* $\Delta(t) = (t, t_{equ_h} \wedge t'_{equ_t}, \mathcal{R}_1{}^*, \infty, \Delta'),\ \Delta' = \Delta_h \cup \Delta'_t$
$\Delta_h \vdash t_{equ_h} \triangleright \mathcal{R}_1$ *s.t.* $\Delta_h(t_{equ_h}) = (t_{equ_h}, t'_{equ_h}, \mathcal{R}_1, \mathcal{S}_{fin_h}, \Delta'_h)$ *for some* $t'_{equ_h}, \mathcal{S}_{fin_h}, \Delta'_h$
$\Delta_t \vdash t_{equ_t} \triangleright \mathcal{R}_1{}^*$ *s.t.* $\Delta_t(t_{equ_t}) = (t_{equ_t}, t'_{equ_t}, \mathcal{R}_1{}^*, \mathcal{S}_{fin_t}, \Delta'_t)$ *for some* $t_{equ_t}, \mathcal{S}_{fin_t}$
satsifies the following conditions
$\Gamma_2 \vdash t_2 : \mathcal{R}_1, \cdots \Gamma_n \vdash t_n : \mathcal{R}_1$ *for some* $\Gamma_2, \cdots \Gamma_n$ *s.t.* $\mathcal{Dom}(\Gamma_j) \cap \mathcal{Dom}(\Gamma_k) = \phi$ *if* $j \neq k$,
and $\bigcup_{j=2}^{n} \mathcal{Dom}(\Gamma_j) \subseteq \mathit{sub}\Big(\mathcal{M}\big(\Delta_t(t_{equ_t})\big)\Big)$

Proof. *Proof is structual induction on derivation of* $\Delta \vdash t \triangleright \mathcal{R}_1{}^*$.

Our assumption of $\Delta(t) = (t, t_{equ_h} \wedge t'_{equ_t}, \mathcal{R}_1{}^*, \infty, \Delta')$ *implies that the last applied rule on* $\Delta \vdash t \triangleright \mathcal{R}_1{}^*$ *is (T-Cat0-canon-∞), thus we know* $t'_{equ_t} \neq \{\}$ *from def. of (T-Cat0-canon-∞). Accrding to the fact of* $\Delta_t \vdash t_{equ_t} \triangleright \mathcal{R}_1{}^*$ *s.t.* $\Delta_t(t_{equ_t}) = (t_{equ_t}, t'_{equ_t}, \mathcal{R}_1{}^*, \mathcal{S}_t, \Delta'_t)$ *with* $t'_{equ_t} \neq \{\}$, *it's enough to consider only 2 cases for the deduction of* $\Delta_t \vdash t_{equ_t} \triangleright \mathcal{R}_1{}^*$ *i.e. the last applied rule on* $\Delta_t \vdash t_{equ_t} \triangleright \mathcal{R}_1{}^*$.

Case (T-Cat0-xtend-sol): We suppose that the last applied rule on $\Delta_t \vdash t_{equ_t} \triangleright \mathcal{R}_1{}^*$ *is (T-Cat0-xtend-sol). In this case, we let* $\Delta_t \vdash t_{equ_t} \triangleright \mathcal{R}_1{}^*$ *s.t.* $\Delta_t = \big\{(t_{equ_t}, t'_{equ_t}, \mathcal{R}_1{}^*, sol, \Delta'_t)\big\} \cup \Delta'_t$, *where* $\Delta'_t \vdash t'_{equ_t} \triangleright \mathcal{R}_1$ *s.t.* $\Delta'_t = \big\{(t'_{equ_t}, t''_{equ_t}, \mathcal{R}_1, \mathcal{S}'_t, \Delta''_t)\big\} \cup \Delta''_t$ *from def. of (T-Cat0-xtend-sol). Then* $\{t_2 \cdots t_n\} = \mathcal{M}_{\{\}}\Big(t'_{equ_t}, \mathcal{R}_1{}^*, \big(\Delta' \setminus \big(\Delta'(t_{equ_h}) \cup \Delta'_h\big)\big)\Big) = \mathcal{M}_{\{\}}(t'_{equ_t}, \mathcal{R}_1{}^*, \Delta'_t) = \Big\{\mathcal{M}\big(\Delta'_t(t'_{equ_t})\big)\Big\}$, *since* $\Big(\Delta' \setminus \big(\Delta'(t_{equ_h}) \cup \Delta'_h\big)\Big) = (\Delta' \setminus \Delta_h) = \Delta'_t$ *where* $\Delta' = (\Delta_h \cup \Delta'_t)$ *s.t.* $\mathcal{Dom}(\Delta_h) \cap \mathcal{Dom}(\Delta'_t) = \phi$, *and* $\Delta'(t_{equ_h}) = (\Delta_h \cup \Delta'_t)(t_{equ_h}) = \Delta_h(t_{equ_h}) = (t_{equ_h}, t'_{equ_h}, \mathcal{R}_1, \mathcal{S}_h, \Delta'_h)$. *Note that* $\Delta_h \vdash t_{equ_h} \triangleright \mathcal{R}_1$ *implies* $t_{equ_h} \in \mathcal{Dom}(\Delta_h)$, *and* $\mathcal{Dom}(\Delta_h) \cap$

$\mathcal{D}om(\Delta'_t) = \phi$ comes from $\mathcal{D}om(\Delta_h) \cap \mathcal{D}om(\Delta_t) = \phi$ with $\mathcal{D}om(\Delta'_t) \subseteq \mathcal{D}om(\Delta_t)$ from def. of (T-Cat0-canon-∞). Applying induction hypothesis on $\Delta'_t \vdash t'_{equ_t} \triangleright \mathcal{R}_1$ brings us $\Gamma'_t \vdash \mathcal{M}\big(\Delta'_t(t'_{equ_t})\big) : \mathcal{R}_1$ for some Γ'_t s.t. $sub\Big(\mathcal{M}\big(\Delta_t(t_{equ_t})\big)\Big) = sub\big(\mathcal{M}(t_{equ_t}, t'_{equ_t}, \mathcal{R}_1{}^*, sol, \Delta'_t)\big) = sub\bigg(\Big\{\mathcal{M}\big(\Delta'_t(t'_{equ_t})\big)\Big\}\bigg) \supseteq sub\Big(\mathcal{M}\big(\Delta'_t(t'_{equ_t})\big)\Big) \supseteq \mathcal{D}om(\Gamma'_t)$ as desired from def. of $\mathcal{M}$ with Property 3.4.

Case (T-Cat0-xtend-∞): We suppose that the last rule applied on the derivation of $\Delta_t \vdash t_{equ_t} \triangleright \mathcal{R}_1{}^*$ is (T-Cat0-canon-∞). Thus we let $\Delta_t = \big\{(t_{equ_t}, t_{equ_{t_h}} \wedge t'_{equ_{t_t}}, \mathcal{R}_1{}^*, \infty, \Delta'_t)\big\} \cup \Delta'_t$ with $\Delta'_t = (\Delta_{t_h} \cup \Delta'_{t_t})$, where we do $\Delta_{t_h} \vdash t_{equ_{t_h}} \triangleright \mathcal{R}_1$ s.t. $\Delta_{t_h} = \big\{(t_{equ_{t_h}}, t'_{equ_{t_h}}, \mathcal{R}_1, \mathcal{S}_{t_h}, \Delta'_{t_h})\big\} \cup \Delta'_{t_h}$ and $\Delta_{t_t} \vdash t_{equ_{t_t}} \triangleright \mathcal{R}_1{}^*$ s.t $\Delta_{t_t} = \big\{(t_{equ_{t_t}}, t'_{equ_{t_t}}, \mathcal{R}_1{}^*, \mathcal{S}_{t_t}, \Delta'_{t_t})\big\} \cup \Delta'_{t_t}$. By applying induction hypothesis on $\Delta_t \vdash t_{equ_t} \triangleright \mathcal{R}_1{}^*$, we obtain $\{t'_2, \cdots t'_m\} = \mathcal{M}_{\{\}}\bigg(t'_{equ_{t_t}}, \mathcal{R}_1{}^*, \Big(\Delta'_t \setminus \big(\Delta'_t(t_{equ_{t_h}}) \cup \Delta'_{t_h}\big)\Big)\bigg)$ s.t. $\Delta'_t = \Delta_{t_h} \cup \Delta'_{t_t}$, which satisfies $\Gamma'_2 \vdash t'_2 : \mathcal{R}_1, \cdots \Gamma'_m \vdash t'_m : \mathcal{R}_1$ for some $\Gamma'_2, \cdots \Gamma'_m$ with $\mathcal{D}om(\Gamma'_j) \cap \mathcal{D}om(\Gamma'_k) = \phi$ where $j \neq k$, and $\bigcup_{j=2}^{m} \mathcal{D}om(\Gamma'_j) \subseteq sub\Big(\mathcal{M}\big(\Delta_{t_t}(t_{equ_{t_t}})\big)\Big)$. Accounting on the facts of $\mathcal{D}om(\Delta'_{t_t}) \subseteq \mathcal{D}om(\Delta_{t_t})$ s.t. $\mathcal{D}om(\Delta_{t_h}) \cap \mathcal{D}om(\Delta_{t_t}) = \phi$ and $t_{equ_{t_h}} \in \mathcal{D}om(\Delta_{t_h})$ s.t. $\Delta_{t_h} \vdash t_{equ_{t_h}} \triangleright \mathcal{R}_1$, we find $t_{equ_{t_h}} \notin \mathcal{D}om(\Delta'_{t_t})$. Consequently $\{t'_2, \cdots t'_m\} = \mathcal{M}_{\{\}}\bigg(t'_{equ_{t_t}}, \mathcal{R}_1{}^*, \Big(\Delta'_t \setminus \big((\Delta_{t_h} \cup \Delta'_{t_t})(t_{equ_{t_h}}) \cup \Delta'_{t_h}\big)\Big)\bigg) = \mathcal{M}_{\{\}}\bigg(t'_{equ_{t_t}}, \mathcal{R}_1{}^*, \Big(\Delta'_t \setminus \big(\Delta_{t_h}(t_{equ_{t_h}}) \cup \Delta'_{t_h}\big)\Big)\bigg) = \mathcal{M}_{\{\}}\Big(t'_{equ_{t_t}}, \mathcal{R}_1{}^*, \big((\Delta_{t_h} \cup \Delta'_{t_t}) \setminus \Delta_{t_h}\big)\Big) = \mathcal{M}_{\{\}}(t'_{equ_{t_t}}, \mathcal{R}_1{}^*, \Delta'_{t_t})$ for $\mathcal{D}om(\Delta_{t_h}) \cap \mathcal{D}om(\Delta'_{t_t}) = \phi$. By now, according to the original judgement of $\Delta \vdash t \triangleright \mathcal{R}_1{}^*$, we consider the term $\{t_2, \cdots t_n\}$ as $\{t_2, \cdots t_n\} = \mathcal{M}_{\{\}}\bigg(t'_{equ_t}, \mathcal{R}_1{}^*, \Big(\Delta' \setminus \big(\Delta'(t_{equ_h}) \cup \Delta'_h\big)\Big)\bigg) = \mathcal{M}_{\{\}}\bigg(t'_{equ_t}, \mathcal{R}_1{}^*, \Big(\Delta' \setminus \big((\Delta_h \cup \Delta'_t)(t_{equ_h}) \cup \Delta'_h\big)\Big)\bigg) = \mathcal{M}_{\{\}}\bigg(t'_{equ_t}, \mathcal{R}_1{}^*, \Big(\Delta' \setminus \big(\Delta_h(t_{equ_h}) \cup \Delta'_h\big)\Big)\bigg) = \mathcal{M}_{\{\}}\Big(t'_{equ_t}, \mathcal{R}_1{}^*, \big((\Delta_h \cup \Delta'_t) \setminus \Delta_h\big)\Big) = \mathcal{M}_{\{\}}(t'_{equ_t}, \mathcal{R}_1{}^*, \Delta'_t) = \mathcal{M}_{\{\}}\big(t'_{equ_t}, \mathcal{R}_1{}^*, (\Delta_{t_h} \cup \Delta'_{t_t})\big) = \mathcal{M}_{\{\}}\big(t_{equ_{t_h}} \wedge t'_{equ_{t_t}}, \mathcal{R}_1{}^*, (\Delta_{t_h} \cup \Delta'_{t_t})\big)$ for $t'_{equ_t} = t_{equ_{t_h}} \wedge t'_{equ_{t_t}}$ and $t_{equ_h} \notin \mathcal{D}om(\Delta'_t)$, which comes from the facts of $\mathcal{D}om(\Delta'_t) \subseteq \mathcal{D}om(\Delta_t)$ s.t. $\mathcal{D}om(\Delta_h) \cap \mathcal{D}om(\Delta_t) = \phi$ with $t_{equ_h} \in \mathcal{D}om(\Delta_h)$ s.t. $\Delta_h \vdash t_{equ_h} : \mathcal{R}_1$. From def. of (T-Cat0-xtend-∞), we know $t_{equ_{t_h}} \wedge t'_{equ_{t_t}} \notin \mathcal{D}om(\Delta_{t_h} \cup \Delta'_{t_t})$, since $\mathcal{D}om(\Delta_{t_h} \cup \Delta'_{t_t}) = \big(\mathcal{D}om(\Delta_{t_h}) \cup \mathcal{D}om(\Delta'_{t_t})\big) \subseteq \big(\mathcal{D}om(\Delta_{t_h}) \cup \mathcal{D}om(\Delta_{t_t})\big) =$

$\mathcal{D}om(\Delta_{t_h} \cup \Delta_{t_t})$ s.t. $t_{equ_{t_h}} \wedge t'_{equ_{t_t}} \notin \mathcal{D}om(\Delta_{t_h} \cup \Delta_{t_t})$ from (T-Cat0-canon-∞). Accordingly we find that $\mathcal{M}_{\{\}}\big(t_{equ_{t_h}} \wedge t'_{equ_{t_t}}, \mathcal{R}_1{}^*, (\Delta_{t_h} \cup \Delta'_{t_t})\big) = \Big\{\mathcal{M}\big(\Delta_{t_h}(t_{equ_{t_h}})\big), t''_2, \cdots t''_l\Big\}$, where $\{t''_2, \cdots t''_l\} = \mathcal{M}_{\{\}}\Big(t'_{equ_{t_t}}, \mathcal{R}_1{}^*, (\Delta_{t_h} \cup \Delta'_{t_t}) \setminus \big((\Delta_{t_h} \cup \Delta'_{t_t})(t_{equ_{t_h}}) \cup \Delta'_{t_h}\big)\Big) = \mathcal{M}_{\{\}}\Big(t'_{equ_{t_t}}, \mathcal{R}_1{}^*, (\Delta_{t_h} \cup \Delta'_{t_t}) \setminus \big(\Delta_{t_h}(t_{equ_{t_h}}) \cup \Delta'_{t_h}\big)\Big) = \mathcal{M}_{\{\}}\big(t'_{equ_{t_t}}, \mathcal{R}_1{}^*, (\Delta_{t_h} \cup \Delta'_{t_t}) \setminus \Delta_{t_h}\big) = \mathcal{M}_{\{\}}(t'_{equ_{t_t}}, \mathcal{R}_1{}^*, \Delta'_{t_t}) = \{t'_2, \cdots t'_m\}$ s.t. $m = l$. Therefore, from the fact we revealed above as $\{t''_2, \cdots t''_l\} = \{t'_2, \cdots t'_m\} = \mathcal{M}_{\{\}}(t'_{equ_{t_t}}, \mathcal{R}_1{}^*, \Delta'_{t_t})$ s.t. $\Gamma'_2 \vdash t'_2 : \mathcal{R}_1, \cdots \Gamma'_m \vdash t'_m : \mathcal{R}_1$ with $\mathcal{D}om(\Gamma'_j) \cap \mathcal{D}om(\Gamma'_k) = \phi$ if $j \neq k$, and $\bigcup_{j=2}^{m} \mathcal{D}om(\Gamma'_j) \subseteq sub\Big(\mathcal{M}\big(\Delta_{t_t}(t_{equ_{t_t}})\big)\Big)$ where $m = l$, we conclude that $\{t_2, \cdots t_n\} = \Big\{\mathcal{M}\big(\Delta_{t_h}(t_{equ_{t_h}})\big), t'_2, \cdots t'_m\Big\}$ satisfies $\Gamma_{t_h} \vdash \mathcal{M}\big(\Delta_{t_h}(t_{equ_{t_h}})\big) : \mathcal{R}_1$ for some Γ_{t_h}, $\Gamma'_2 \vdash t'_2 : \mathcal{R}_1, \cdots \Gamma'_m \vdash t'_m : \mathcal{R}_1$ by applying induction hypothesis on $\Delta_{t_h} \vdash t_{equ_{t_h}} \triangleright \mathcal{R}_1$, with $\mathcal{D}om(\Gamma_{t_h}) \cap \mathcal{D}om(\Gamma'_j) = \phi$ where $2 \leq j \leq m$ for $\mathcal{D}om(\Gamma_{t_h}) \cap \bigcup_{j=2}^{m} \mathcal{D}om(\Gamma'_j) = \phi$, since $\bigcup_{j=2}^{m} \mathcal{D}om(\Gamma'_j) \subseteq sub\Big(\mathcal{M}\big(\Delta_{t_t}(t_{equ_{t_t}})\big)\Big)$ s.t. $sub\Big(\mathcal{M}\big(\Delta_{t_h}(t_{equ_{t_h}})\big)\Big) \cap sub\Big(\mathcal{M}\big(\Delta_{t_t}(t_{equ_{t_t}})\big)\Big) = \phi$ satisfies $\mathcal{D}om(\Gamma_{t_h}) \subseteq sub\Big(\mathcal{M}\big(\Delta_{t_h}(t_{equ_{t_h}})\big)\Big)$ and $\bigcup_{j=2}^{m} \mathcal{D}om(\Gamma'_j) \subseteq sub\Big(\mathcal{M}\big(\Delta_{t_t}(t_{equ_{t_t}})\big)\Big)$ respectively, from Property 7.2 and 3.4 applied on the facts of $\Delta_{t_h} \vdash t_{equ_{t_h}} \triangleright \mathcal{R}_1$ and $\Delta_{t_t} \vdash t_{equ_{t_t}} \triangleright \mathcal{R}_1{}^*$ with the constraint of $\mathcal{D}om(\Delta_{t_h}) \cap \mathcal{D}om(\Delta_{t_t}) = \phi$. Then $\big(\mathcal{D}om(\Gamma_{t_h}) \cup \bigcup_{j=2}^{m} \mathcal{D}om(\Gamma'_j)\big) \subseteq sub\Big(\mathcal{M}\big(\Delta_t(t_{equ_t})\big)\Big)$, since $sub\Big(\mathcal{M}\big(\Delta_t(t_{equ_t})\big)\Big) = sub\big(\mathcal{M}(t_{equ_t}, t_{equ_{t_h}} \wedge t'_{equ_{t_t}}, \mathcal{R}_1{}^*, \infty, \Delta'_t)\big) = sub\Big(\mathcal{M}\big(t_{equ_t}, t_{equ_{t_h}} \wedge t'_{equ_{t_t}}, \mathcal{R}_1{}^*, \infty, (\Delta_{t_h} \cup \Delta'_{t_t})\big)\Big) = sub\bigg(\Big\{\mathcal{M}\big(\Delta_{t_h}(t_{equ_{t_h}})\big), t'_2, \cdots t'_m\Big\}\bigg) \supseteq sub\Big(\mathcal{M}\big(\Delta_{t_h}(t_{equ_{t_h}})\big)\Big) \cup \bigcup_{j=2}^{m} sub(t'_j) \supseteq \mathcal{D}om(\Gamma_{t_h}) \cup \bigcup_{j=2}^{m} \mathcal{D}om(\Gamma'_j)$ for $\Gamma_{t_h} \vdash \mathcal{M}\big(\Delta_{t_h}(t_{equ_{t_h}})\big) : \mathcal{R}_1$, as desired. $\square$

8 Appendix 2: LIezA internal

Prototype of LIezA source codes written in Caml, experimental implementation.

lie_main.ml:

```
(*
  #load "lie_type.cmo";;
  #load "lie_equiv.cmo";;
  #load "lie_lexer.cmo";;
  #load "lie_parser.cmo";;
  #load "lie_main.cmo";;
  open Lie_type;;
  open Lie_equiv;;
  open Lie_main;;
 *)


open List
open Printf
open Lie_type
open Lie_parser
open Lie_lexer
open Lie_trans




let pat_addr_initial = 1;;
let ter_addr_initial = 1;;


let carve_pat pat =
  let rec carving p n =
    match p with
      Pat_ent (op, id, _) -> (Pat_ent (op, id, n), n + 1)
    | Pat_una (op, p1, _) -> (match (carving p1 n) with
                                (p1', n') -> (Pat_una (op, p1', n'), n' + 1) )
    | Pat_bin (op, pl, pr, _) -> (match (carving pr n) with
                                    (pr', n') -> (match (carving pl n') with
                                                    (pl', n'') -> (Pat_bin (op, pl', pr',n'), n' + 1) )
                                  )
  in
  match (carving pat pat_addr_initial) with
    (carved_pat, m) -> carved_pat;;


let carve_term ter =
  let rec carving t n =
    match t with
      Term_ent (op, id, sp, _) -> (Term_ent (op, id, sp, n), n + 1)
    | Term_una (op, t1) -> (match (carving t1 n) with
                              (t1', n') -> (Term_una (op, t1'), n'))
    | Term_bin (op, tl, tr) -> (match (carving tr n) with
                                  (tr', n') -> (match (carving tl n') with
                                                  (tl', n'') -> (Term_bin (op, tl', tr'), n''))
                                )
  in
  match (carving ter ter_addr_initial) with
    (carved_term, m) -> carved_term;;




let cli src =
  try
    let pat = Lie_parser.main Lie_lexer.token (Lexing.from_string src) in
    Some pat
  with
    (Lex_err msg) -> printf "%s.\n" msg;
                     None;;
let compile src =
  match (cli src) with
    Some CLI (ter, pat) -> Some (CLI( (carve_term ter), pat ))
  | None -> None;;


let cli_ter cli =
  match cli with
```

```
    Some CLI (ter, pat) -> ter;;


let cli_pat cli =
  match cli with
    Some CLI (ter, pat) -> pat;;


let compile_ter src_str =
  cli_ter (compile (src_str ^ ": R*"));;


let compile_pat pat_str =
  cli_pat (compile ("{} :" ^ pat_str));;




let rec discomp_ter ter =
  let rec tail_cat0 t =
    match t with
      Term_bin (STAR, th, ts) -> (discomp_ter th) ^ "," ^ (tail_cat0 ts)
    | _ -> (discomp_ter t)
  in
  let rec tail_cat1 t =
    match t with
      Term_bin (CROSS, th, ts) -> (discomp_ter th) ^ "," ^ (tail_cat1 ts)
    | _ -> (discomp_ter t)
  in
  let rec tail_dup t =
    match t with
      Term_bin (STROK, th, ts) -> (discomp_ter th) ^ "," ^ (tail_dup ts)
    | _ -> (discomp_ter t)
  in
  match ter with
    Term_ent (op, id, sp, ad) -> id ^ (if (sp <> "") then  ("_" ^ sp) else "")
  | Term_una (op, t1) -> (match op with
                            STAR -> (match t1 with
                                       Term_ent (NIL, _, _, _) -> "{}"
                                     | _ -> "{" ^ (discomp_ter t1) ^ "}" )
                          | CROSS -> "[" ^ (discomp_ter t1) ^ "]"
                          | STROK -> "<" ^ (discomp_ter t1) ^ ">"
                          | OPT -> (match t1 with
                                      Term_ent (NIL, _, _, _) -> "*"
                                    | _ -> (discomp_ter t1) ^ "?" )
                          | LEFT -> (discomp_ter t1) ^ "<-"
                          | RIGHT -> (discomp_ter t1) ^ "->"
                          | _ -> raise (Illegal_ter_detected (ter, __LINE__, __FILE__))
                         )
  | Term_bin (STAR, th, ts) -> "{" ^ (tail_cat0 ter) ^ "}"
  | Term_bin (CROSS, th, ts) -> "{" ^ (tail_cat1 ter) ^ "}"
  | Term_bin (STROK, th, ts) -> "{" ^ (tail_dup ter) ^ "}"
  | Term_bin (WEDGE, tl, tr) -> (match tl with
                                   Term_ent (ENT, id1, sp1, ad1) ->
                                    (match tr with
                                       Term_ent (ENT, _, _, _) -> "(" ^ (discomp_ter tl) ^ " & " ^ (discomp_ter tr) ^ ")"
                                     | Term_una (_, t1) -> "(" ^ (discomp_ter tl) ^ " & " ^ (discomp_ter tr) ^ ")"
                                     | Term_bin (_, trl, trr) -> "(" ^ (discomp_ter tl) ^ " & (" ^ (discomp_ter tr) ^ "))"
                                     | _ -> raise (Illegal_ter_detected (tr, __LINE__, __FILE__)) )
                                 | Term_una (_, tl1) ->
                                    (match tr with
                                       Term_ent (ENT, _, _, _) -> "(" ^ (discomp_ter tl) ^ " & " ^ (discomp_ter tr) ^ ")"
                                     | Term_una (_, t1) -> "(" ^ (discomp_ter tl) ^ " & " ^ (discomp_ter tr) ^ ")"
                                     | Term_bin (_, trl, trr) -> "(" ^ (discomp_ter tl) ^ " & (" ^ (discomp_ter tr) ^ "))"
                                     | _ -> raise (Illegal_ter_detected (tr, __LINE__, __FILE__)) )
                                 | Term_bin (_, tll, tlr) ->
                                    (match tr with
                                       Term_ent (ENT, _, _, _) -> "((" ^ (discomp_ter tl) ^ ") & " ^ (discomp_ter tr) ^ ")"
                                     | Term_una (_, t1) -> "((" ^ (discomp_ter tl) ^ ") & " ^ (discomp_ter tr) ^ ")"
                                     | Term_bin (_, trl, trr) -> "((" ^ (discomp_ter tl) ^ ") & (" ^ (discomp_ter tr) ^ "))"
                                     | _ -> raise (Illegal_ter_detected (tr, __LINE__, __FILE__)) )
                                 | _ -> raise (Illegal_ter_detected (tl, __LINE__, __FILE__))
                                )
  | Term_bin (VEE, tl, tr) -> (match tl with
                                 Term_ent (ENT, id1, sp1, ad1) ->
                                  (match tr with
                                     Term_ent (ENT, _, _, _) -> "(" ^ (discomp_ter tl) ^ " | " ^ (discomp_ter tr) ^ ")"
                                   | Term_una (_, t1) -> "(" ^ (discomp_ter tl) ^ " | " ^ (discomp_ter tr) ^ ")"
                                   | Term_bin (_, trl, trr) -> "(" ^ (discomp_ter tl) ^ " | (" ^ (discomp_ter tr) ^ "))"
                                   | _ -> raise (Illegal_ter_detected (tr, __LINE__, __FILE__)) )
                               | Term_una (_, tl1) ->
                                  (match tr with
                                     Term_ent (ENT, _, _, _) -> "(" ^ (discomp_ter tl) ^ " | " ^ (discomp_ter tr) ^ ")"
                                   | Term_una (_, t1) -> "(" ^ (discomp_ter tl) ^ " | " ^ (discomp_ter tr) ^ ")"
```

```
                                    | Term_bin (_, trl, trr) -> "(" ^ (discomp_ter tl) ^ " | (" ^ (discomp_ter tr) ^ "))"
                                    | _ -> raise (Illegal_ter_detected (tr, __LINE__, __FILE__)) )
                                | Term_bin (_, tll, tlr) ->
                                   (match tr with
                                      Term_ent (ENT, _, _, _) -> "((" ^ (discomp_ter tl) ^ ") | " ^ (discomp_ter tr) ^ ")"
                                    | Term_una (_, t1) -> "((" ^ (discomp_ter tl) ^ ") | " ^ (discomp_ter tr) ^ ")"
                                    | Term_bin (_, trl, trr) -> "((" ^ (discomp_ter tl) ^ ") | (" ^ (discomp_ter tr) ^ "))"
                                    | _ -> raise (Illegal_ter_detected (tr, __LINE__, __FILE__)) )
                                | _ -> raise (Illegal_ter_detected (tl, __LINE__, __FILE__))
                               )
  | _ -> raise (Illegal_ter_detected (ter, __LINE__, __FILE__));;


let term0 t =
  discomp_ter t;;


let term1 t =
  match t with
    Some t -> term0 t
  | None -> "";;


let foreach ts =
  match ts with
    None -> []
  | Some tl -> map term0 tl;;


let term judgement =
  match judgement with
    Some {ter = t} -> term0 t
  | None -> "";;


let cterm judgement =
  match (canon judgement) with
    Some c_t -> term0 c_t
  | None -> "";;
```

lie_type.ml:

```
type term_ope =
  | NIL
  | ENT
  | STAR
  | CROSS
  | STROK
  | OPT
  | ALT
  | LEFT
  | RIGHT
  | WEDGE
  | VEE
  | SEQ;;

type pattern =
  | Pat_ent of term_ope * string * int
  | Pat_una of term_ope * pattern * int
  | Pat_bin of term_ope * pattern * pattern * int

type term =
  | Term_ent of term_ope * string * string * int
  | Term_una of term_ope * term
  | Term_bin of term_ope * term * term;;

type cli =
  | CLI of term * pattern;;


type fin_sym =
  | FIN_GND
  | FIN_DEF
  | FIN_WEDGE
  | FIN_VEE
  | FIN_NIL
  | FIN_SOL
  | FIN_INFTY
  | FIN_L
  | FIN_R;;

type binding =
  {ter:term; equ:term; pat:pattern; fin:fin_sym; bindings:binding list};;



exception Illegal_pat_detected of pattern * int * string;;
exception Illformed_bindings_detected of term * pattern * int * string;;
exception Illformed_equterm_detected of term * pattern * int * string;;
exception Illegal_ter_detected of term * int * string;;



let rec term_size t =
  match t with
    Term_ent (NIL, "", sp, ad) -> 0
  | Term_ent (ENT, id, sp, ad) -> 1
  | Term_una (STAR, t_1) ->(term_size t_1) + 1
  | Term_bin (STAR, t_h, t_tl) -> (match t_tl with
                                     Term_bin (STAR, _, _) -> (term_size t_h) + (term_size t_tl)
                                   | _ -> (term_size t_h) + (term_size t_tl) + 1 )
  | Term_una (CROSS, t_1) ->(term_size t_1) + 1
  | Term_bin (CROSS, t_h, t_tl) -> (match t_tl with
                                      Term_bin (CROSS, _, _) -> (term_size t_h) + (term_size t_tl)
                                    | _ -> (term_size t_h) + (term_size t_tl) + 1 )
  | Term_una (STROK, t_1) ->(term_size t_1) + 1
  | Term_bin (STROK, t_h, t_tl) -> (match t_tl with
                                      Term_bin (STROK, _, _) -> (term_size t_h) + (term_size t_tl)
                                    | _ -> (term_size t_h) + (term_size t_tl) + 1 )
  | Term_una (OPT, t_1) -> (term_size t_1) + 1
  | Term_una (LEFT, t_1) -> (term_size t_1) + 1
  | Term_una (RIGHT, t_1) -> (term_size t_1) + 1
  | Term_bin (WEDGE, t_1, t_2) -> (term_size t_1) + (term_size t_2) + 1
  | Term_bin (VEE, t_1, t_2) -> (term_size t_1) + (term_size t_2) + 1
  | _ -> raise (Illegal_ter_detected (t, __LINE__, __FILE__));;


let rec pat_size p =
  match p with
  | Pat_ent (NIL, "", ad) -> 0
  | Pat_ent (ENT, id, ad) -> 1
  | Pat_bin (WEDGE, p_1, p_2, ad) -> (pat_size p_1) + (pat_size p_2) + 1
  | Pat_bin (VEE, p_1, p_2, ad) -> (pat_size p_1) + (pat_size p_2) + 1
  | Pat_una (STAR, p_1, ad) -> (pat_size p_1) + 1
  | Pat_una (CROSS, p_1, ad) -> (pat_size p_1) + 1
  | Pat_una (STROK, p_1, ad) -> (pat_size p_1) + 1
  | Pat_una (OPT, p_1, ad) -> (pat_size p_1) + 1
  | Pat_bin (ALT, p_L, p_R, ad) -> (pat_size p_L) + (pat_size p_R) + 1
  | _ -> raise (Illegal_pat_detected (p, __LINE__, __FILE__));;


let rec set_nelems tl =
  match tl with
    [] -> 0
  | (t::ts) -> (set_nelems ts) + 1


let rec pat_ident p1 p2 =
  match p1 with
    Pat_ent (op1, id1, ad1) -> (match p2 with
                                  Pat_ent (op2, id2, ad2) -> ((op1 = op2) && (id1 = id2))
                                | _ -> false )
  | Pat_una (op1, p1_pri, ad1) -> (match p2 with
                                     Pat_una (op2, p2_pri, ad2) -> ((op1 = op2) && (pat_ident p1_pri p2_pri))
                                   | _ -> false )
  | Pat_bin (op1, p1_l, p1_r, ad1) -> (match p2 with
                                         Pat_bin (op2, p2_l, p2_r, ad2) -> ((op1 = op2) && (pat_ident p1_l p2_l) && (pat_ident p1_r p2_r))
                                       | _ -> false )


let rec term_ident t1 t2 =
  match t1 with
    Term_ent (op1, id1, sp1, ad1) -> (match t2 with
                                        Term_ent (op2, id2, sp2, ad2) -> ((op1 = op2) && (id1 = id2) && (sp1 = sp2) && (ad1 = ad2))
                                      | _ -> false )
  | Term_una (op1, t1_pri) -> (match t2 with
                                 Term_una (op2, t2_pri) -> ((op1 = op2) && (term_ident t1_pri t2_pri))
                               | _ -> false )
  | Term_bin (op1, t1_l, t1_r) -> (match t2 with
                                     Term_bin (op2, t2_l, t2_r) ->
                                      ((op1 == op2) && (term_ident t1_l t2_l) && (term_ident t1_r t2_r))
                                   | _ -> false )


let rec set_sub s1 s2 =
  let rec rid t hd tl =
    match tl with
      [] -> hd
    | (x::xs) -> if (term_ident x t) then (hd @ xs) else (rid t (hd @ [x]) xs)
  in
  match s1 with
    [] -> s2
  | (x::xs) -> set_sub xs (rid x [] s2)


let rec set_union det s1 s2 =
  let add t tl =
    let rec exists t tl =
      match tl with
        [] -> false
      | x::xs -> if (det x t) then true else (exists t xs)
    in
    if (exists t tl) then tl else (t::tl)
  in
  match s1 with
    [] -> s2
  | x::xs -> set_union det xs (add x s2);;
```

lie_equiv.ml:

```
open Lie_type


type assoc_dir =
  | L2R
  | R2L;;


(* Brings next equivalent term over associativity on cascade connectivity. *)
let rec equiv_assoc_cas (t, dir) =
  match t with
    Term_ent (op, id, sp, ad) -> (None, dir)
  | Term_una (op, t1) -> (None, dir)
  | Term_bin (WEDGE, Term_ent (op_l, id_l, sp_l, ad_l), Term_ent (op_r, id_r, sp_r, ad_r)) -> (None, dir)
  | Term_bin (WEDGE, Term_ent (op_l, id_l, sp_l, ad_l), Term_una (op_r, t_r)) -> (None, dir)
  | Term_bin (WEDGE, Term_una (op_l, t_l), Term_ent (op_r, id_r, sp_r, ad_r)) -> (None, dir)
  | Term_bin (WEDGE, Term_una (op_l, t_l), Term_una (op_r, t_r)) -> (None, dir)
  | Term_bin (WEDGE, l, r) -> assoc_cas_family l r dir
  | Term_bin (_, l, r) -> (None, dir)
and assoc_cas_family l r d =
  let assoc_cas_R l r =
    let rec sft2_r l r =
      match l with
        Term_ent (op, id, sp, ad) -> None
      | Term_una (op, l1) -> None
      | Term_bin (WEDGE, ll, Term_ent (op_lr, id_lr, sp_lr, ad_lr)) -> Some (ll, Term_bin (WEDGE, Term_ent (op_lr, id_lr, sp_lr, ad_lr), r))
      | Term_bin (WEDGE, ll, Term_una (op_lr, lr1)) -> Some (ll, Term_bin (WEDGE, Term_una (op_lr, lr1), r))
      | Term_bin (WEDGE, ll, lr) -> (match lr with
                                      Term_bin (WEDGE, _, _) -> (match (sft2_r lr r) with
                                                                    None -> None
                                                                  | Some (pl, pr) -> Some (Term_bin (WEDGE, ll, pl), pr) )
                                    | _ -> Some (ll, Term_bin (WEDGE, lr, r)) )
      | Term_bin (_, ll, lr) -> None
    in
    match (sft2_r l r) with
      None -> None
    | Some (l’, r’) -> Some (Term_bin (WEDGE, l’, r’))
  in
  let assoc_cas_L l r =
    let rec sft2_l l r =
      match r with
        Term_ent (op, id, sp, ad) -> None
      | Term_una (op, r1) -> None
      | Term_bin (WEDGE, Term_ent (op_rl, id_rl, sp_rl, ad_rl), rr) -> Some (Term_bin (WEDGE, l, Term_ent (op_rl, id_rl, sp_rl, ad_rl)), rr)
      | Term_bin (WEDGE, Term_una (op_rl, rl1), rr) -> Some (Term_bin (WEDGE, l, Term_una (op_rl, rl1)), rr)
      | Term_bin (WEDGE, rl, rr) -> (match rl with
                                      Term_bin (WEDGE, _, _) -> (match (sft2_l l rl) with
                                                                    None -> None
                                                                  | Some (pl, pr) -> Some (pl, Term_bin (WEDGE, pr, rr)) )
                                    | _ -> Some (Term_bin (WEDGE, l, rl), rr) )
      | Term_bin (_, rl, rr) -> None
    in
    match (sft2_l l r) with
      None -> None
    | Some (l’, r’) -> Some (Term_bin (WEDGE, l’, r’))
  in
  match d with
    L2R -> (match l with
              Term_bin (WEDGE, ll, lr) -> ((assoc_cas_R l r), L2R)
            | _ -> (None, L2R) )
  | R2L -> (match r with
              Term_bin (WEDGE, rl, rr) -> ((assoc_cas_L l r), R2L)
            | _ -> (None, R2L) );;




let rec equiv_assoc_par (t, dir) =
  match t with
    Term_ent (op, id, sp, ad) -> (None, dir)
  | Term_una (op, t1) -> (None, dir)
  | Term_bin (VEE, Term_ent (op_l, id_l, sp_l, ad_l), Term_ent (op_r, id_r, sp_r, ad_r)) -> (None, dir)
  | Term_bin (VEE, Term_ent (op_l, id_l, sp_l, ad_l), Term_una (op_r, t_r)) -> (None, dir)
  | Term_bin (VEE, Term_una (op_l, t_l), Term_ent (op_r, id_r, sp_r, ad_r)) -> (None, dir)
  | Term_bin (VEE, Term_una (op_l, t_l), Term_una (op_r, t_r)) -> (None, dir)
  | Term_bin (VEE, l, r) -> assoc_par_family l r dir
  | Term_bin (_, l, r) -> (None, dir)
and assoc_par_family l r d =
  let assoc_par_R l r =
    let rec sft2_r l r =
      match l with
        Term_ent (op, id, sp, ad) -> None
      | Term_una (op, ll) -> None
      | Term_bin (VEE, ll, Term_ent (op_lr, id_lr, sp_lr, ad_lr)) -> Some (ll, Term_bin (VEE, Term_ent (op_lr, id_lr, sp_lr, ad_lr), r))
      | Term_bin (VEE, ll, Term_una (op_lr, lr1)) -> Some (ll, Term_bin (VEE, Term_una (op_lr, lr1), r))
      | Term_bin (VEE, ll, lr) -> (match lr with
                                      Term_bin (VEE, _, _) -> (match (sft2_r lr r) with
                                                                  None -> None
                                                                | Some (pl, pr) -> Some (Term_bin (VEE, ll, pl), pr) )
                                    | _ -> Some (ll, Term_bin (VEE, lr, r)) )
      | Term_bin (_, ll, lr) -> None
    in
    match (sft2_r l r) with
      None -> None
    | Some (l', r') -> Some (Term_bin (VEE, l', r'))
  in
  let assoc_par_L l r =
    let rec sft2_l l r =
      match r with
        Term_ent (op, id, sp, ad) -> None
      | Term_una (op, rl) -> None
      | Term_bin (VEE, Term_ent (op_rl, id_rl, sp_rl, ad_rl), rr) -> Some (Term_bin (VEE, l, Term_ent (op_rl, id_rl, sp_rl, ad_rl)), rr)
      | Term_bin (VEE, Term_una (op_rl, rl1), rr) -> Some (Term_bin (VEE, l, Term_una (op_rl, rl1)), rr)
      | Term_bin (VEE, rl, rr) -> (match rl with
                                      Term_bin (VEE, _, _) -> (match (sft2_l l rl) with
                                                                  None -> None
                                                                | Some (pl, pr) -> Some (pl, Term_bin (VEE, pr, rr)) )
                                    | _ -> Some (Term_bin (VEE, l, rl), rr) )
      | Term_bin (_, rl, rr) -> None
    in
    match (sft2_l l r) with
      None -> None
    | Some (l', r') -> Some (Term_bin (VEE, l', r'))
  in
  match d with
    L2R -> (match l with
              Term_bin (VEE, ll, lr) -> ((assoc_par_R l r), L2R)
            | _ -> (None, L2R) )
  | R2L -> (match r with
              Term_bin (VEE, rl, rr) -> ((assoc_par_L l r), R2L)
            | _ -> (None, R2L) );;



(* gathering by "Associativity-Cas" and "Associativity-Par" *)
let equiv_assocs (t_orig, t_resume, assoc_dir) =
  match t_resume with
    Term_ent (op, id, sp, ad) -> (None, assoc_dir)
  | Term_una (op, t1) -> (None, assoc_dir)
  | Term_bin (WEDGE, l, r) -> let t_res' = (equiv_assoc_cas (t_resume, assoc_dir)) in
                              (match t_res' with
                                 (None, R2L) -> (None, R2L)
                               | (None, L2R) -> equiv_assoc_cas (t_orig, R2L)
                               | (Some e, _) -> t_res' )
  | Term_bin (VEE, l, r) -> let t_res' = (equiv_assoc_par (t_resume, assoc_dir)) in
                              (match t_res' with
                                 (None, R2L) -> (None, R2L)
                               | (None, L2R) -> equiv_assoc_par (t_orig, R2L)
                               | (Some e, _) -> t_res' )
  | Term_bin (_, l, r) -> (None, assoc_dir);;



let rec equiv_terms (t_orig, t_resume, assoc_dir) ena_bumpup =
  let derive t =
    let equ_ph2 = (gath_equivs t) in
    let equ_ph3 = (gath_equ_ph3 equ_ph2) in
    let equ_ph4 = if ena_bumpup then (gath_equ_ph4 equ_ph2) else equ_ph2
    in
    (set_union term_ident equ_ph2 (set_union term_ident equ_ph3 equ_ph4))
  in
  match t_resume with
    None -> (Some t_orig, (derive t_orig), L2R)
  | Some t_res -> let equ_assoc = (equiv_assocs (t_orig, t_res, assoc_dir))
                  in
                  match equ_assoc with
                    (None, dir) -> (None, [], dir)
                  | (Some e, dir) -> (Some e, (derive e), dir)


and gath_equivs t =
  (* gathering equivalents by "Identity" and
     "Associativity-Cat0_1", "Associativity-Cat1_1", "Associativity-Dup_1",
     "AssociativityH-Cat0_infty", "AssociativityH-Cat1_infty", "AssociativityH-Dup_infty",
     "Optional", "Alt-L", "Alt-R". *)
  let rec gath_equ_ph2 equiv =
    let strip_grp t =
      match t with
        Term_una (STAR, t1) -> (match t1 with
                                   Term_ent (NIL, "", "", ad) -> []
                                 | _ -> [t1])
      | Term_una (CROSS, t1) -> [t1]
      | Term_una (STROK, t1) -> [t1]
      | Term_una (OPT, t1) -> (match t1 with
                                  Term_ent (NIL, "", "", ad) -> []
                                | _ -> [t1])
      | Term_una (LEFT, t1) -> [t1]
      | Term_una (RIGHT, t1) -> [t1]
      | _ -> []
    in
    let split_grp t =
      match t with
      | Term_bin (STAR, tl, tr) -> let tr' = (match tr with
                                                Term_ent (op, id, sp, ad) -> Term_una (STAR, tr)
                                              | _ -> tr)
                                   in
                                   [Term_bin (WEDGE, tl, tr')]
      | Term_bin (CROSS, tl, tr) -> let tr' = (match tr with
                                                 Term_ent (op, id, sp, ad) -> Term_una (CROSS, tr)
                                               | _ -> tr)
                                    in
                                    [Term_bin (WEDGE, tl, tr')]
      | Term_bin (STROK, tl, tr) -> let tr' = (match tr with
                                                 Term_ent (op, id, sp, ad) -> Term_una (STROK, tr)
                                               | _ -> tr)
                                    in
                                    [Term_bin (VEE, tl, tr')]
      | _ -> []
    in
    match equiv with
    | Term_ent (op, id, sp, ad) -> [equiv]
    | Term_una (op, t1) ->  [equiv] @ (strip_grp equiv)
    | Term_bin (WEDGE, tl, tr) -> [equiv]
    | Term_bin (VEE, tl, tr) -> [equiv]
    | Term_bin (_, _, _) -> [equiv] @ (split_grp equiv)
  in
  gath_equ_ph2 t


(* gathering by "RevealH-Cas_{}", "RevealT-Cas_{}", "RevealH-Par_{}", "RevealT-Par_{}" and
   "RevealH-Cas_o", "RevealT-Cas_o", "RevealH-Par_o", "RevealT-Par_o". *)
and gath_equ_ph3 equivs =
  match equivs with
    [] -> []
  | (e::es) -> (match e with
                  Term_ent (op, id, sp, ad) -> e
                | Term_una (op, t1) -> e
                | Term_bin (WEDGE, Term_una ( STAR, Term_ent (NIL, "", "", ad) ), t_r) -> t_r
                | Term_bin (WEDGE, t_l, Term_una ( STAR, Term_ent (NIL, "", "", ad) )) -> t_l
                | Term_bin (VEE, Term_una ( STAR, Term_ent (NIL, "", "", ad) ), t_r) -> t_r
                | Term_bin (VEE, t_l, Term_una ( STAR, Term_ent (NIL, "", "", ad) )) -> t_l
                | Term_bin (WEDGE, Term_una ( OPT, Term_ent (NIL, "", "", ad) ), t_r) -> t_r
                | Term_bin (WEDGE, t_l, Term_una ( OPT, Term_ent (NIL, "", "", ad) )) -> t_l
                | Term_bin (VEE, Term_una ( OPT, Term_ent (NIL, "", "", ad) ), t_r) -> t_r
                | Term_bin (VEE, t_l, Term_una ( OPT, Term_ent (NIL, "", "", ad) )) -> t_l
                | Term_bin (_, _, _) -> e ) :: (gath_equ_ph3 es)


(* gathering by "PhonyH-Cas_{}", "PhonyT-Cas_{}", "PhonyH-Par_{}", "PhonyT-Par_{}" and
   "PhonyH-Cas_o", "PhonyT-Cas_o", "PhonyH-Par_o", "PhonyT-Par_o".  *)
and gath_equ_ph4 equivs =
  let rec max_addr_term ter =
    match ter with
      Term_ent (_, _, _, ad) -> ad
    | Term_una (_, t1) -> max_addr_term t1
    | Term_bin (_, tl, tr) -> let max_l = (max_addr_term tl) in
                              let max_r = (max_addr_term tr)
                              in
                              if (max_l > max_r) then max_l else max_r
  in
  match equivs with
    [] -> []
  | (e::es) ->
     let bumped_ones = (match e with
                          Term_bin (WEDGE, Term_una( STAR, Term_ent (NIL, "", "", ad) ), _) -> e
                        | _ -> Term_bin (WEDGE, Term_una( STAR, Term_ent (NIL, "", "", (max_addr_term e) + 1) ), e))
                       :: (match e with
                             Term_bin (WEDGE, _, Term_una( STAR, Term_ent (NIL, "", "", ad) )) -> e
                           | _ -> Term_bin (WEDGE, e, Term_una( STAR, Term_ent (NIL, "", "", (max_addr_term e) + 1)) ))
                       :: (match e with
                             Term_bin (VEE, Term_una( STAR, Term_ent (NIL, "", "", ad) ), _) -> e
                           | _ -> Term_bin (VEE, Term_una( STAR, Term_ent (NIL, "", "", (max_addr_term e) + 1) ), e))
                       :: (match e with
                             Term_bin (VEE, _, Term_una( STAR, Term_ent (NIL, "", "", ad) )) -> e
                           | _ -> Term_bin (VEE, e, Term_una( STAR, Term_ent (NIL, "", "", (max_addr_term e) + 1)) ))
                       :: (match e with
                             Term_bin (WEDGE, Term_una( OPT, Term_ent (NIL, "", "", ad) ), _) -> e
                           | _ -> Term_bin (WEDGE, Term_una( OPT, Term_ent (NIL, "", "", (max_addr_term e) + 1) ), e))
                       :: (match e with
                             Term_bin (WEDGE, _, Term_una( OPT, Term_ent (NIL, "", "", ad) )) -> e
                           | _ -> Term_bin (WEDGE, e, Term_una( OPT, Term_ent (NIL, "", "", (max_addr_term e) + 1)) ))
                       :: (match e with
                             Term_bin (VEE, Term_una( OPT, Term_ent (NIL, "", "", ad) ), _) -> e
                           | _ -> Term_bin (VEE, Term_una( OPT, Term_ent (NIL, "", "", (max_addr_term e) + 1) ), e))
                       :: (match e with
                             Term_bin (VEE, _, Term_una( OPT, Term_ent (NIL, "", "", ad) )) -> [e]
                           | _ -> [Term_bin (VEE, e, Term_una( OPT, Term_ent (NIL, "", "", (max_addr_term e) + 1) ))])
     in
     set_union term_ident bumped_ones (gath_equ_ph4 es);;



let rec  merge_wedge tr_l tr_r =
  let rec attach l tr_r =
    match tr_r with
      [] -> []
    | r::rs -> (Term_bin (WEDGE, l, r))::(attach l rs)
  in
  match tr_l with
    [] -> []
  | l::ls -> (attach l tr_r) @ (merge_wedge ls tr_r);;


let rec merge_vee tr_l tr_r =
  let rec attach l tr_r =
    match tr_r with
      [] -> []
    | r::rs -> (Term_bin (VEE, l, r))::(attach l rs)
  in
  match tr_l with
    [] -> []
  | l::ls -> (attach l tr_r) @ (merge_vee ls tr_r);;
```

lie_match.ml:

```
open Lie_type
open Lie_equiv



let rec idx_lkup pat idx =
  let addr pat =
    match pat with
      Pat_ent (_, _, ad) -> ad
    | Pat_una (_, _, ad) -> ad
    | Pat_bin (_, _, _, ad) -> ad
  in
  match idx with
    [] -> -1
  | i::is -> match i with
             (ad, prefix_size) -> if (ad = (addr pat)) then prefix_size
                                  else (idx_lkup pat is);;


(* fetches "equivs" the new equivalent set derived from next equivalent term over associativity,
   and tries to infer with given rule "cmp" for each equivalents. *)
let boost cmp (ter_orig, ena_bumpup) pat idx =
  let oracle p t = (term_size t) >= (idx_lkup p idx)
  in
  let rec revolver cmp (ter_orig, ter_crnt, assoc_dir, ena_bumpup) pat =
    if (oracle pat ter_orig) then
      let equivs = (equiv_terms (ter_orig, ter_crnt, assoc_dir) ena_bumpup) in
      match equivs with
        (None, _, _) -> None
      | (Some ter', e_ts, dir) -> match (cmp e_ts pat) with
                                    Some found -> Some found;
                                  | None -> revolver cmp (ter_orig, (Some ter'), dir, ena_bumpup) pat
    else None
  in
  revolver cmp (ter_orig, None, L2R, ena_bumpup) pat;;


let binds_union bindings1 bindings2 =
  match bindings1 with
    []-> (match bindings2 with
            [] -> []
          | bs2 -> bs2)
  | bs1 -> match bindings2 with
             [] -> bs1
           | bs2 -> (bs1 @ bs2);;


(* namely, null predicator with slight inteligence. *)
let rec is_nil t =
  match t with
    Term_ent (NIL, id, sp, ad) -> Some t
  | Term_una (op, t1) ->
     (match op with
        STAR -> (match (is_nil t1) with
                   Some (Term_ent (NIL, id, sp, ad)) -> Some (Term_una ( STAR, (Term_ent (NIL, id, sp, ad)) ))
                 | Some t1' -> Some t1'
                 | None -> None)
      | CROSS -> is_nil t1
      | STROK -> is_nil t1
      | OPT -> (match (is_nil t1) with
                  Some (Term_ent (NIL, id, sp, ad)) -> Some (Term_una ( OPT, (Term_ent (NIL, id, sp, ad)) ))
                | Some t1' -> Some t1'
                | None -> None)
      | LEFT -> is_nil t1
      | RIGHT -> is_nil t1
      | _ -> None)
  | Term_bin (op, t_l, t_r) ->
     if ((op == WEDGE) || (op == VEE)) then
       (match (is_nil t_l) with
          None -> None
        | Some nil_l -> (match (is_nil t_r) with
                           None -> None
                         | Some nil_r -> Some nil_r) )
     else None
  | _ -> None;;


(* matching engine core, ITS THE COMPLEX OF WISDOM. *)
let rec tourbillon ter pat idx =
  (* Decides the rule to be applied for inference according to syntax of given pattern. *)
  match pat with
    (* for the case of "t_Atom0_impl" *)
    Pat_ent (ENT, id, ad) -> (match (t_Atom0_impl ter pat idx) with
                                None -> None
                              | Some judge_matched -> Some judge_matched)
  (* for the case of "t_Cas_impl" *)
  | Pat_bin (WEDGE, p_1, p_2, ad) -> (match (t_Cas_impl ter pat idx) with
                                        None -> None
                                      | Some judge_matched -> Some judge_matched)
  (* for the case of "t_Par_impl" *)
  | Pat_bin (VEE, p_1, p_2, ad) -> (match (t_Par_impl ter pat idx) with
                                      None -> None
                                    | Some judge_matched -> Some judge_matched)
  (* for the case of "t_Cat0_impl_nil" *)
  | Pat_una (STAR, p_1, ad) -> (match (t_Cat0_impl_nil ter pat) with
                                  Some judge_matched -> Some judge_matched
                                | None -> (match (t_Cat0_impl_sol ter p_1 idx) with
                                             Some judge_matched -> Some judge_matched
                                           |  None -> (match (t_Cat0_impl_infty ter p_1 idx) with
                                                         None -> None
                                                       | Some judge_matched -> Some judge_matched) ) )
  (* for the case of "t_Cat1_impl_sol" *)
  | Pat_una (CROSS, p_1, ad) -> (match (t_Cat1_impl_sol ter p_1 idx) with
                                   Some judge_matched -> Some judge_matched
                                 |  None -> (match (t_Cat1_impl_infty ter p_1 idx) with
                                               None -> None
                                             | Some judge_matched -> Some judge_matched) )
  (* for the case of "t_Dup_impl_sol" *)
  | Pat_una (STROK, p_1, ad) -> (match (t_Dup_impl_sol ter p_1 idx) with
                                   Some judge_matched -> Some judge_matched
                                 |  None -> (match (t_Dup_impl_infty ter p_1 idx) with
                                               None -> None
                                             | Some judge_matched -> Some judge_matched) )
  (* for the case of "t_Opt_xtend_nil" *)
  | Pat_una (OPT, p_1, ad) -> (match (t_Opt_xtend_nil ter pat idx) with
                                 Some judge_matched -> Some judge_matched
                               | None -> (match (t_Opt_impl_sol ter p_1 idx) with
                                            None -> None
                                          | Some judge_matched -> Some judge_matched) )
  (* for the case of "t_Alt_impl" *)
  | Pat_bin (ALT, p_L, p_R, ad) -> (match (t_Alt_impl ter pat idx) with
                                      None -> None
                                    | Some judge_matched -> Some judge_matched)
  | _ -> raise (Illegal_pat_detected (pat, __LINE__, __FILE__))


and t_Atom0_impl ter pat idx =
  let rec cmp_atomic t pat =
    match pat with
      Pat_ent (ENT, p_id, p_ad) ->
       (match t with
          Term_ent (ENT, t_id, t_sp, t_ad) -> if (t_id = p_id) then Some t else None
        | _ -> None)
    | _ -> raise (Illegal_pat_detected (pat, __LINE__, __FILE__))
  and match_atomic tl pat =
    match tl with
      [] -> None
    | (x::xs) -> match (cmp_atomic x pat) with
                   Some found -> Some {ter = ter; equ = found; pat = pat; fin = FIN_GND; bindings = []}
                 | None -> match_atomic xs pat
  in
  boost match_atomic (ter, false) pat idx


and t_Cas_impl ter pat idx =
  let rec cmp_cat t pat =
    match pat with
      Pat_bin (WEDGE, p_1, p_2, ad) ->
       (match t with
          Term_bin (WEDGE, t_l, t_r) ->
           let r_1st = (tourbillon t_l p_1 idx)
           in
           (match r_1st with
            | None -> None
            | Some b_1st -> let r_2nd = (tourbillon t_r p_2 idx)
                            in
                            (match r_2nd with
                             | None -> None
                             | Some b_2nd ->
                                 let bindings' = (binds_union [b_1st] [b_2nd])
                                 in
                                 (match bindings' with
                                    [] -> raise (Illformed_bindings_detected (t, pat, __LINE__, __FILE__))
                                  | b -> let e = Term_bin (WEDGE, b_1st.ter, b_2nd.ter) in
                                         Some {ter = ter; equ = e; pat = pat; fin = FIN_WEDGE; bindings = b} )
                            )
          )
        | _ -> None
       )
    | _ -> raise (Illegal_pat_detected (pat, __LINE__, __FILE__))
  and match_cat tl pat =
    match tl with
      [] -> None
    | (x::xs) -> match (cmp_cat x pat) with
                   Some found -> Some found
                 | None -> match_cat xs pat
  in
  boost match_cat (ter, true) pat idx


and t_Par_impl ter pat idx =
  let rec cmp_par t pat =
    match pat with
      Pat_bin (VEE, p_1, p_2, ad) ->
       (match t with
          Term_bin (VEE, t_l, t_r) ->
           let r_1st = (tourbillon t_l p_1 idx)
           in
           (match r_1st with
            | None -> None
            | Some b_1st -> let r_2nd = (tourbillon t_r p_2 idx)
                            in
                            (match r_2nd with
                             | None -> None
                             | Some b_2nd ->
                                let bindings' = (binds_union [b_1st] [b_2nd])
                                in
                                (match bindings' with
                                   [] -> raise (Illformed_bindings_detected (t, pat, __LINE__, __FILE__))
                                 | b -> let e = Term_bin (VEE, b_1st.ter, b_2nd.ter) in
                                        Some {ter = ter; equ = e; pat = pat; fin = FIN_VEE; bindings = b} )
                            )
          )
        | _ -> None
       )
    | _ -> raise (Illegal_pat_detected (pat, __LINE__, __FILE__))
  and match_par tl pat =
    match tl with
      [] -> None
    | (x::xs) -> match (cmp_par x pat) with
                   Some found -> Some found
                 | None -> match_par xs pat
  in
  boost match_par (ter, true) pat idx


and t_Cat0_impl_nil ter pat =
  match (is_nil ter) with
    None -> None
  | Some v ->
     let v' = (match v with
                 Term_una ( STAR, Term_ent (NIL, id, sp, ad) ) -> v
               | Term_una ( OPT, Term_ent (NIL, id, sp, ad) ) -> Term_una ( STAR, Term_ent (NIL, id, sp, ad) )
               | _ -> raise (Illformed_equterm_detected (v, pat, __LINE__, __FILE__)) )
     in
     Some {ter = ter; equ = v'; pat = pat; fin = FIN_NIL; bindings = []}


and t_Cat0_impl_sol ter pat idx =
  let rec match_sol tl pat =
    match tl with
      [] -> None
    | (x::xs) -> match (tourbillon x pat idx) with
                   Some found -> Some {ter = ter; equ = found.ter; pat = (Pat_una (STAR, found.pat, -1)); fin = FIN_SOL; bindings = [found]}
                 | None -> match_sol xs pat
  in
  boost match_sol (ter, true) pat idx


and t_Cat0_impl_infty ter pat idx =
  let disbumping ter =
    match ter with
      Term_bin (WEDGE, Term_una ( STAR, Term_ent (NIL, "", "", ad) ), t_t) ->
       (match t_t with
          Term_bin (WEDGE, t_t_h, t_t_t) -> Some (t_t_h, t_t_t)
        | _ -> None)
    | Term_bin (WEDGE, t_h, t_t) -> Some (t_h, t_t)
    | _ -> None
  in
  let rec cmp_cat0 t pat =
    match (disbumping t) with
      Some (t_h, t_t) ->
       let r_h = (tourbillon t_h pat idx)
       in
       (match r_h with
        | None -> None
        | Some b_h -> let r_t = (tourbillon t_t (Pat_una (STAR, pat, -1)) idx)
                      in
                      (match r_t with
                       | None -> None
                       | Some b_t -> let bindings' = (binds_union [b_h] b_t.bindings)
                                     in
                                     (match bindings' with
                                        [] -> raise (Illformed_bindings_detected (t, pat, __LINE__, __FILE__))
                                      | b -> let e = Term_bin (WEDGE, b_h.ter, b_t.equ)
                                             in
                                             Some {ter = ter; equ = e; pat = Pat_una (STAR, pat, -1); fin = FIN_INFTY; bindings = b} )
                      )
       )
    | _ -> None
  and match_cat0 tl pat =
    match tl with
      [] -> None
    | (x::xs) -> match (cmp_cat0 x pat) with
                   Some found -> Some found
                 | None -> match_cat0 xs pat
  in
  boost match_cat0 (ter, true) pat idx


and t_Cat1_impl_sol ter pat idx =
  let rec match_sol tl pat =
    match tl with
      [] -> None
    | (x::xs) -> match (tourbillon x pat idx) with
                   Some found -> Some {ter = ter; equ = found.ter; pat = (Pat_una (CROSS, found.pat, -1)); fin = FIN_SOL; bindings = [found]}
                 | None -> match_sol xs pat
  in
  boost match_sol (ter, true) pat idx


and t_Cat1_impl_infty ter pat idx =
  let disbumping ter =
    match ter with
      Term_bin (WEDGE, Term_una ( STAR, Term_ent (NIL, "", "", ad) ), t_t) ->
       (match t_t with
          Term_bin (WEDGE, t_t_h, t_t_t) -> Some (t_t_h, t_t_t)
        | _ -> None)
    | Term_bin (WEDGE, t_h, t_t) -> Some (t_h, t_t)
    | _ -> None
  in
  let rec cmp_cat1 t pat =
    match (disbumping t) with
      Some (t_h, t_t) ->
       let r_h = (tourbillon t_h pat idx)
       in
       (match r_h with
        | None -> None
        | Some b_h -> let r_t = (tourbillon t_t (Pat_una (CROSS, pat, -1)) idx)
                      in
                      (match r_t with
                       | None -> None
                       | Some b_t -> let bindings' = (binds_union [b_h] b_t.bindings)
                                     in
                                     (match bindings' with
                                        [] -> raise (Illformed_bindings_detected (t, pat, __LINE__, __FILE__))
                                      | b -> let e = Term_bin (WEDGE, b_h.ter, b_t.equ)
                                             in
                                             Some {ter = ter; equ = e; pat = Pat_una (CROSS, pat, -1); fin = FIN_INFTY; bindings = b}
                                     )
                      )
       )
    | _ -> None
  and match_cat1 tl pat =
    match tl with
      [] -> None
    | (x::xs) -> match (cmp_cat1 x pat) with
                   Some found -> Some found
                 | None -> match_cat1 xs pat
  in
  boost match_cat1 (ter, true) pat idx


and t_Dup_impl_sol ter pat idx =
  let rec match_sol tl pat =
    match tl with
      [] -> None
    | (x::xs) -> match (tourbillon x pat idx) with
                   Some found -> Some {ter = ter; equ = found.ter; pat = (Pat_una (STROK, found.pat, -1)); fin = FIN_SOL; bindings = [found]}
                 | None -> match_sol xs pat
  in
  boost match_sol (ter, true) pat idx


and t_Dup_impl_infty ter pat idx =
  let disbumping ter =
    match ter with
      Term_bin (VEE, Term_una ( STAR, Term_ent (NIL, "", "", ad) ), t_t) ->
       (match t_t with
          Term_bin (VEE, t_t_h, t_t_t) -> Some (t_t_h, t_t_t)
        | _ -> None)
    | Term_bin (VEE, t_h, t_t) -> Some (t_h, t_t)
    | _ -> None
  in
  let rec cmp_dup t pat =
    match (disbumping t) with
      Some (t_h, t_t) ->
       let r_h = (tourbillon t_h pat idx)
       in
       (match r_h with
        | None -> None
        | Some b_h -> let r_t = (tourbillon t_t (Pat_una (STROK, pat, -1)) idx)
                      in
                      (match r_t with
                       | None -> None
                       | Some b_t -> let bindings' = (binds_union [b_h] b_t.bindings)
                                     in
                                     (match bindings' with
                                        [] -> raise (Illformed_bindings_detected (t, pat, __LINE__, __FILE__))
                                      | b -> let e = Term_bin (VEE, b_h.ter, b_t.equ)
                                             in
                                             Some {ter = ter; equ = e; pat = Pat_una (STROK, pat, -1); fin = FIN_INFTY; bindings = b}
                                     )
                      )
       )
    | _ -> None
  and match_dup tl pat =
    match tl with
      [] -> None
    | (x::xs) -> match (cmp_dup x pat) with
                   Some found -> Some found
                 | None -> match_dup xs pat
  in
  boost match_dup (ter, true) pat idx


and t_Opt_xtend_nil ter pat idx =
  match (is_nil ter) with
    None -> None
  | Some v -> let v' = (match v with
                        | Term_una ( STAR, Term_ent (NIL, id, sp, ad) ) -> Term_una ( OPT, Term_ent (NIL, id, sp, ad) )
                        | Term_una ( OPT, Term_ent (NIL, id, sp, ad) ) -> v
                        | _ -> raise (Illformed_equterm_detected (v, pat, __LINE__, __FILE__)) )
              in
              Some {ter = ter; equ = v'; pat = pat; fin = FIN_NIL; bindings = []}


and t_Opt_impl_sol ter pat idx =
  let rec match_sol tl pat =
    match tl with
      [] -> None
    | (x::xs) -> match (tourbillon x pat idx) with
                   Some found -> Some {ter = ter; equ = found.ter; pat = (Pat_una (OPT, found.pat, -1)); fin = FIN_SOL; bindings = [found]}
                 | None -> match_sol xs pat
  in
  boost match_sol (ter, true) pat idx


and t_Alt_impl ter pat idx =
  let rec cmp_alt t pat =
    match pat with
      Pat_bin (ALT, p_L, p_R, ad) ->
       let r_L = (tourbillon t p_L idx)
       in
       (match r_L with
          Some b_L -> Some {ter = ter; equ = b_L.ter; pat = pat; fin = FIN_L; bindings = [b_L]}
        | None -> let r_R = (tourbillon t p_R idx)
                  in
                  (match r_R with
                   | Some b_R -> Some {ter = t; equ = b_R.ter; pat = pat; fin = FIN_R; bindings = [b_R]}
                   | None -> None
                  )
       )
    | _ -> raise (Illegal_pat_detected (pat, __LINE__, __FILE__))
  and match_alt tl pat =
    match tl with
      [] -> None
    | (x::xs) -> match (cmp_alt x pat) with
                   Some found -> Some found
                 | None -> match_alt xs pat
  in
  boost match_alt (ter, true) pat idx;;




let first pat =
  let rec gath_prefix p =
    let is_nil prefix = pat_ident prefix (Pat_ent (NIL, "", -1))
    in
    match p with
      Pat_ent (ENT, id, ad) -> p
    | Pat_bin (WEDGE, p_1, p_2, ad) -> let fir_1 = (gath_prefix p_1) in
                                       let fir_2 = (gath_prefix p_2) in
                                       if (is_nil fir_1) then
                                         if (is_nil fir_2) then Pat_ent (NIL, "", -1) else fir_2
                                       else
                                         if (is_nil fir_2) then fir_1 else Pat_bin (WEDGE, fir_1, fir_2, -1)
    | Pat_bin (VEE, p_1, p_2, ad) -> let fir_1 = (gath_prefix p_1) in
                                     let fir_2 = (gath_prefix p_2) in
                                     if (is_nil fir_1) then
                                       if (is_nil fir_2) then Pat_ent (NIL, "", -1) else fir_2
                                     else
                                       if (is_nil fir_2) then fir_1 else Pat_bin (VEE, fir_1, fir_2, -1)
    | Pat_una (STAR, p_1, ad) -> Pat_ent (NIL, "", -1)
    | Pat_una (CROSS, p_1, ad) -> gath_prefix p_1
    | Pat_una (STROK, p_1, ad) -> gath_prefix p_1
    | Pat_una (OPT, p_1, ad) -> Pat_ent (NIL, "", -1)
    | Pat_bin (ALT, p_L, p_R, ad) -> let fir_L = (gath_prefix p_L) in
                                     let fir_R = (gath_prefix p_R) in
                                     if (pat_size fir_L) >= (pat_size fir_R) then fir_L else fir_R
    | _ -> raise (Illegal_pat_detected (pat, __LINE__, __FILE__))
  in
  gath_prefix pat;;


let rec perf_index pat =
  match pat with
    Pat_ent (ENT, id, ad) -> [(ad, pat_size (first pat))]
  | Pat_bin (WEDGE, p_1, p_2, ad) -> (perf_index p_1) @ (perf_index p_2) @ [(ad, pat_size (first pat))]
  | Pat_bin (VEE, p_1, p_2, ad) -> (perf_index p_1) @ (perf_index p_2) @ [(ad, pat_size (first pat))]
  | Pat_una (STAR, p_1, ad) -> (perf_index p_1) @ [(ad, pat_size (first pat))]
  | Pat_una (CROSS, p_1, ad) -> (perf_index p_1) @ [(ad, pat_size (first pat))]
  | Pat_una (STROK, p_1, ad) -> (perf_index p_1) @ [(ad, pat_size (first pat))]
  | Pat_una (OPT, p_1, ad) -> (perf_index p_1) @ [(ad, pat_size (first pat))]
  | Pat_bin (ALT, p_1, p_2, ad) -> (perf_index p_1) @ (perf_index p_2) @ [(ad, pat_size (first pat))]
  | _ -> raise (Illegal_pat_detected (pat, __LINE__, __FILE__));;


let rec curve pat ad =
  match pat with
    Pat_ent (ENT, id, _) -> (Pat_ent (ENT, id, ad), ad)
  | Pat_bin (WEDGE, p_1, p_2, _) -> let r1 = (curve p_1 ad) in
                                    (match r1 with
                                       (p_1’, ad_1’) -> let r2 = (curve p_2 (ad_1’ + 1)) in
                                                        (match r2 with
                                                           (p_2’, ad_2’) -> (Pat_bin (WEDGE, p_1’, p_2’, (ad_2’ + 1)), (ad_2’ + 1)) )
                                    )
  | Pat_bin (VEE, p_1, p_2, _) -> let r1 = (curve p_1 ad) in
                                    (match r1 with
                                       (p_1’, ad_1’) -> let r2 = (curve p_2 (ad_1’ + 1)) in
                                                        (match r2 with
                                                           (p_2’, ad_2’) -> (Pat_bin (VEE, p_1’, p_2’, (ad_2’ + 1)), (ad_2’ + 1)) )
                                    )
  | Pat_una (STAR, p_1, _) -> (match (curve p_1 ad) with
                                 (p_1’, ad’) -> (Pat_una (STAR, p_1’, (ad’ + 1)), (ad’ + 1)) )
  | Pat_una (CROSS, p_1, _) -> (match (curve p_1 ad) with
                                  (p_1’, ad’) -> (Pat_una (CROSS, p_1’, (ad’ + 1)), (ad’ + 1)) )
  | Pat_una (STROK, p_1, _) -> (match (curve p_1 ad) with
                                  (p_1’, ad’) -> (Pat_una (STROK, p_1’, (ad’ + 1)), (ad’ + 1)) )
  | Pat_una (OPT, p_1, _) -> (match (curve p_1 ad) with
                                (p_1’, ad’) -> (Pat_una (OPT, p_1’, (ad’ + 1)), (ad’ + 1)) )
  | Pat_bin (ALT, p_L, p_R, _) -> let rL = (curve p_L ad) in
                                  (match rL with
                                     (p_L’, ad_L’) -> let rR = (curve p_R (ad_L’ + 1)) in
                                                      (match rR with
                                                         (p_R’, ad_R’) -> (Pat_bin (ALT, p_L’, p_R’, (ad_R’ + 1)), (ad_R’ + 1)) )
                                  )
  | _ -> raise (Illegal_pat_detected (pat, __LINE__, __FILE__));;


let typematch cli =
  match cli with
    None -> None
  | Some CLI (ter, pat) -> match (curve pat 1) with
                             (pat’, _) -> tourbillon ter pat’ (perf_index pat’);;
```

lie_trans.ml:

```
open Lie_type


let rec bindings_sub bindings1 bindings2 =
  let rec rid b hd bl =
    let ident b1 b2 =
      match b1 with
        {ter = t_1} -> match b2 with
                         {ter = t_2} -> (t_1 = t_2)
    in
    match bl with
      [] -> hd
    | (x::xs) -> if (ident x b) then (hd @ xs) else (rid b (hd @ [x]) xs)
  in
  match bindings1 with
    [] -> bindings2
  | (b::bs) -> bindings_sub bs (rid b [] bindings2);;


let lkup_bindings ter bindings =
  let rec descend ter binding =
    match binding with
      {ter = t; bindings = bindings'} -> if (term_ident t ter) then (Some binding)
                                        else (traverse ter bindings')
  and traverse ter bindings =
    match bindings with
      [] -> None
    | (b::bs) -> let found = (descend ter b)
                 in
                 match found with
                   Some binding -> found
                 | None -> (traverse ter bs)
  in
  traverse ter bindings;;


let rec synthesize binding =
  match binding with
    {ter = t; equ = t_e; pat = p; fin = fin_sym; bindings = bindings'} ->
    (match fin_sym with
       FIN_GND -> t_e
     | FIN_DEF -> t_e
     | FIN_NIL -> t_e
     | FIN_WEDGE -> (match t_e with
                       Term_bin (WEDGE, e_l, e_r) -> resolv_bin WEDGE t (e_l, e_r) p bindings'
                     | _ -> raise (Illformed_equterm_detected (t_e, p, __LINE__, __FILE__)) )
     | FIN_VEE -> (match t_e with
                     Term_bin (VEE, e_l, e_r) -> resolv_bin VEE t (e_l, e_r) p bindings'
                   | _ -> raise (Illformed_equterm_detected (t_e, p, __LINE__, __FILE__)) )
     | FIN_SOL -> let t1's_binding = (lkup_bindings t_e bindings')
                  in (match t1's_binding with
                        Some b_1 -> synthesize b_1
                      | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
     | FIN_INFTY -> (match t_e with
                       Term_bin (WEDGE, e_h, e_t') -> resolv_n'ary WEDGE t (e_h, e_t') p bindings'
                     | Term_bin (VEE, e_h, e_t') -> resolv_n'ary VEE t (e_h, e_t') p bindings'
                     | _ -> raise (Illformed_equterm_detected (t_e, p, __LINE__, __FILE__)) )
     | FIN_L -> let t1's_binding = (lkup_bindings t_e bindings')
                in (match t1's_binding with
                      Some b_l -> synthesize b_l
                    | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
     | FIN_R -> let t1's_binding = (lkup_bindings t_e bindings')
                in (match t1's_binding with
                      Some b_r -> synthesize b_r
                    | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
    )


and resolv_bin op ter (e_l, e_r) pat bindings' =
  let e_l's_binding = (lkup_bindings e_l bindings') in
  let e_r's_binding = (lkup_bindings e_r bindings')
  in
  match e_l's_binding with
    Some b_l -> (match e_r's_binding with
                   Some b_r -> Term_bin (op, (synthesize b_l), (synthesize b_r))
                 | None -> raise (Illformed_bindings_detected (ter, pat, __LINE__, __FILE__)) )
  | None -> raise (Illformed_bindings_detected (ter, pat, __LINE__, __FILE__))


and resolv_n'ary op ter (e_l, e_r) pat bindings' =
  let e_l's_binding = (lkup_bindings e_l bindings')
  in
  match e_l's_binding with
    Some b_l ->
     let bindings'' = (bindings_sub [b_l] bindings')
     in (match bindings'' with
           (b_t'::bs) -> (match bs with
                            [] -> Term_bin (op, (synthesize b_l), (synthesize b_t'))
                          | b_t_t'' ->
                             let b_t = {ter = e_r; equ = e_r; pat = pat; fin = FIN_INFTY; bindings = bindings''}
                             in (Term_bin (op, (synthesize b_l), (synthesize b_t))) )
         | _ -> raise (Illformed_bindings_detected (ter, pat, __LINE__, __FILE__))
        )
  | None -> raise (Illformed_bindings_detected (ter, pat, __LINE__, __FILE__));;


let matched_form judgement =
  match judgement with
    None -> None
  | Some binding -> Some [(synthesize binding)];;


let rec resolv pat judgement =
  let combine r_h r_tl =
    match r_h with
      None -> r_tl
    | Some ts_h -> (match r_tl with
                      None -> r_h
                    | Some ts_tl -> Some (ts_h @ ts_tl) )
  in
  let rec walk bl =
    match bl with
      [] -> None
    | (b::bs) -> combine (resolv pat (Some b)) (walk bs)
  in
  match judgement with
    None -> None
  | Some binding -> match binding with
                      {ter = t; pat = p; bindings = bindings'} ->
                      if (pat_ident pat p) then (matched_form (Some binding))
                      else
                        (match p with
                           Pat_ent (ENT, id, ad) -> None
                         | Pat_bin (WEDGE, p_1, p_2, ad) ->
                            (match bindings' with
                               (b_1::b_2::[]) -> let r_1 = (resolv pat (Some b_1))
                                                 in (match r_1 with
                                                       None -> (resolv pat (Some b_2))
                                                     | Some ts_1 -> (match (resolv pat (Some b_2)) with
                                                                       None -> r_1
                                                                     | Some ts_2 -> Some (ts_1 @ ts_2) )
                                                    )
                             | _ -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
                         | Pat_bin (VEE, p_1, p_2, ad) ->
                            (match bindings' with
                               (b_1::b_2::[]) -> let r_1 = (resolv pat (Some b_1))
                                                 in (match r_1 with
                                                       None -> (resolv pat (Some b_2))
                                                     | Some ts_1 -> (match (resolv pat (Some b_2)) with
                                                                       None -> r_1
                                                                     | Some ts_2 -> Some (ts_1 @ ts_2) )
                                                    )
                             | _ -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
                         | Pat_una (STAR, p_1, ad) -> walk bindings'
                         | Pat_una (CROSS, p_1, ad) -> walk bindings'
                         | Pat_una (STROK, p_1, ad) -> walk bindings'
                         | Pat_una (OPT, p_1, ad) ->
                            (match bindings' with
                               [] -> None
                             | (b_opt::[]) -> resolv pat (Some b_opt)
                             | _ -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
                         | Pat_bin (ALT, p_L, p_R, ad) ->
                            (match bindings' with
                               (b_alt::[]) -> resolv pat (Some b_alt)
                             | _ -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
                         | _ -> raise (Illegal_pat_detected (p, __LINE__, __FILE__))
                        );;


let rec canonicalize judgement =
  match judgement with
    {ter = t; equ = t_e; pat = p; fin = fin_sym; bindings = bindings'} ->
    (match fin_sym with
       FIN_GND -> t_e
     | FIN_DEF -> t_e
     | FIN_NIL -> t_e
     | FIN_WEDGE -> (match t_e with
                       Term_bin (WEDGE, e_1, e_2) ->
                        (match (lkup_bindings e_1 bindings’) with
                           Some b_1 -> let c_1 = (canonicalize b_1)
                                       in
                                       (match (lkup_bindings e_2 bindings’) with
                                          Some b_2 -> let c_2 = (canonicalize b_2) in Term_bin (WEDGE, c_1, c_2)
                                        | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__))
                                       )
                         | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__))
                        )
                     | _ -> raise (Illformed_equterm_detected (t_e, p, __LINE__, __FILE__))
                    )
     | FIN_VEE -> (match t_e with
                     Term_bin (VEE, e_1, e_2) ->
                      (match (lkup_bindings e_1 bindings’) with
                         Some b_1 -> let c_1 = (canonicalize b_1)
                                     in
                                     (match (lkup_bindings e_2 bindings’) with
                                        Some b_2 -> let c_2 = (canonicalize b_2) in Term_bin (VEE, c_1, c_2)
                                      | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__))
                                     )
                       | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__))
                      )
                   | _ -> raise (Illformed_equterm_detected (t_e, p, __LINE__, __FILE__))
                  )
     | FIN_SOL -> (match p with
                     Pat_una (STAR, p1, ad) -> (match (lkup_bindings t_e bindings’) with
                                                  Some b -> Term_una (STAR, (canonicalize b))
                                                | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
                   | Pat_una (CROSS, p1, ad) -> (match (lkup_bindings t_e bindings’) with
                                                   Some b -> Term_una (CROSS, (canonicalize b))
                                                 | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
                   | Pat_una (STROK, p1, ad) -> (match (lkup_bindings t_e bindings’) with
                                                   Some b -> Term_una (STROK, (canonicalize b))
                                                 | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
                   | Pat_una (OPT, p1, ad) -> (match (lkup_bindings t_e bindings’) with
                                                 Some b -> Term_una (OPT, (canonicalize b))
                                               | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
                   | _ -> raise (Illegal_pat_detected (p, __LINE__, __FILE__))
                  )
     | FIN_INFTY -> (match p with
                       Pat_una (STAR, p1, ad) ->
                        (match t_e with
                           Term_bin (WEDGE, e_h, e_t) ->
                            (match (lkup_bindings e_h bindings’) with
                               Some b_h -> (match (lkup_bindings e_t bindings’) with
                                              Some b_t -> Term_bin ( STAR, (canonicalize b_h), (canonicalize b_t) )
                                            | None -> (match (bindings_sub [b_h] bindings’) with
                                                          [] -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__))
                                                       | b_ts -> Term_bin ( STAR, (canonicalize b_h), (canon_cat0 t e_t p b_ts) ) )

                                           )
                             | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
                         | _ -> raise (Illformed_equterm_detected (t_e, p, __LINE__, __FILE__))
                        )
                     | Pat_una (CROSS, p1, ad) ->
                        (match t_e with
                           Term_bin (WEDGE, e_h, e_t) ->
                            (match (lkup_bindings e_h bindings’) with
                               Some b_h -> (match (lkup_bindings e_t bindings’) with
                                              Some b_t -> Term_bin ( CROSS, (canonicalize b_h), (canonicalize b_t) )
                                            | None -> (match (bindings_sub [b_h] bindings’) with
                                                          [] -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__))
                                                        | b_ts -> Term_bin ( CROSS, (canonicalize b_h), (canon_cat1 t e_t p b_ts) ) )
                                           )
                             | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
                         | _ -> raise (Illformed_equterm_detected (t_e, p, __LINE__, __FILE__))
                        )
                     | Pat_una (STROK, p1, ad) ->
                        (match t_e with
                           Term_bin (VEE, e_h, e_t) ->
                            (match (lkup_bindings e_h bindings’) with
                               Some b_h -> (match (lkup_bindings e_t bindings’) with
                                              Some b_t -> Term_bin ( STROK, (canonicalize b_h), (canonicalize b_t) )
                                            | None -> (match (bindings_sub [b_h] bindings’) with
                                                          [] -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__))
                                                        | b_ts -> Term_bin ( STROK, (canonicalize b_h), (canon_dup t e_t p b_ts) ) )
                                           )
                             | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
                         | _ -> raise (Illformed_equterm_detected (t_e, p, __LINE__, __FILE__))
                        )
                     | _ -> raise (Illegal_pat_detected (p, __LINE__, __FILE__))
                    )
```

```
     | FIN_L -> (match (lkup_bindings t_e bindings') with
                  Some b -> Term_una (LEFT, (canonicalize b))
                | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
     | FIN_R -> (match (lkup_bindings t_e bindings') with
                  Some b -> Term_una (RIGHT, (canonicalize b))
                | None -> raise (Illformed_bindings_detected (t, p, __LINE__, __FILE__)) )
    )


and canon_cat0 t_orig t p b =
  match (lkup_bindings t b) with
    Some b' -> Term_una ( STAR, (canonicalize b') )
  | None -> (match t with
               Term_bin (WEDGE, t_h, t_t) ->
                (match (lkup_bindings t_h b) with
                   Some b_h -> (match (bindings_sub [b_h] b) with
                                  [] -> raise (Illformed_bindings_detected (t_orig, p, __LINE__, __FILE__))
                                | b_ts -> let c_tl = (canon_cat0 t_orig t_t p b_ts)
                                          in
                                          (match c_tl with
                                             Term_una (STAR, tl) -> Term_bin (STAR, (canonicalize b_h), tl)
                                           | _ -> Term_bin (STAR, (canonicalize b_h), c_tl) )
                               )
                 | None -> Term_una ( STAR, t )
                )
             | _ -> Term_una ( STAR, t )
            )


and canon_cat1 t_orig t p b =
  match (lkup_bindings t b) with
    Some b' -> Term_una ( CROSS, (canonicalize b') )
  | None -> (match t with
               Term_bin (WEDGE, t_h, t_t) ->
                (match (lkup_bindings t_h b) with
                   Some b_h -> (match (bindings_sub [b_h] b) with
                                  [] -> raise (Illformed_bindings_detected (t_orig, p, __LINE__, __FILE__))
                                | b_ts -> let c_tl = (canon_cat1 t_orig t_t p b_ts)
                                          in
                                          (match c_tl with
                                             Term_una (CROSS, tl) -> Term_bin (CROSS, (canonicalize b_h), tl)
                                           | _ -> Term_bin (CROSS, (canonicalize b_h), c_tl) )
                               )
                 | None -> Term_una ( CROSS, t )
                )
             | _ -> Term_una ( CROSS, t )
            )


and canon_dup t_orig t p b =
  match (lkup_bindings t b) with
    Some b' -> Term_una ( STROK, (canonicalize b') )
  | None -> (match t with
               Term_bin (VEE, t_h, t_t) ->
                (match (lkup_bindings t_h b) with
                   Some b_h -> (match (bindings_sub [b_h] b) with
                                  [] -> raise (Illformed_bindings_detected (t_orig, p, __LINE__, __FILE__))
                                | b_ts -> let c_tl = (canon_dup t_orig t_t p b_ts)
                                          in
                                          (match c_tl with
                                             Term_una (STROK, tl) -> Term_bin (STROK, (canonicalize b_h), tl)
                                           | _ -> Term_bin (STROK, (canonicalize b_h), c_tl) )
                               )
                 | None -> Term_una ( STROK, t )
                )
             | _ -> Term_una ( STROK, t )
            );;



let canon judgement =
  match judgement with
    Some j -> Some (canonicalize j)
  | None  -> None;;
```

lie_lexer.mll

```
{
  open Lie_parser
  exception Lex_err of string
}
(* lexer definition *)
let space = [' ' '\t' '\n' '\r']
let digit = ['0'-'9']
let alpha = ['A'-'Z' 'a'-'z']
let alnum = alpha | digit | ['#']

rule token = parse
  (* operators *)
  | "<-" { LEFT }
  | "->" { RIGHT }
  | '&' { WEDGE }
  | '|' { VEE }
  | '*' { STAR }
  | '+' { CROSS }
  | '?' { OPT }
  | '!' { STROK }
  | '%' { ALT }
  | '(' { LPAR }
  | ')' { RPAR }
  | '{' { LBRA }
  | '}' { RBRA }
  | '[' { LSQB }
  | ']' { RSQB }
  | '<' { LBIL }
  | '>' { RBIL }
  | ',' { COMMA }
  | ':' { COLON }
  | '_' { SPECIFIER }

  (* entities *)
  | alpha alnum* { IDENT1 (Lexing.lexeme lexbuf) }
  | alnum+ {IDENT0 (Lexing.lexeme lexbuf) }
  | space+ { token lexbuf }
  | eof { EOI }
  | _ {
        let msg = Printf.sprintf
                    "unknown token %s near characters %d-%d"
                    (Lexing.lexeme lexbuf)
                    (Lexing.lexeme_start lexbuf)
                    (Lexing.lexeme_end lexbuf)
        in
        raise (Lex_err msg)
      }
```

lie_parser.mly:

```
%{
open Lie_type
%}
%token <string> IDENT0
%token <string> IDENT1
%token LBRA RBRA
%token LSQB RSQB
%token LBIL RBIL
%token COMMA
%token LPAR RPAR

%Token COLON
%token SPECIFIER

%token STAR CROSS STROK OPT
%token ALT
%token LEFT
%token RIGHT
%token WEDGE VEE
%token EOI

%right WEDGE VEE
%right ALT
%nonassoc STAR CROSS STROK OPT
%nonassoc LEFT RIGHT

%start main
%type <Lie_type.cli> main

%%
main:
  expr_term COLON expr_pat EOI    { CLI ($1, $3) }
;

expr_term:
  IDENT1    { Term_ent (ENT, $1, "", -1 ) }
| IDENT1 SPECIFIER IDENT0    { Term_ent (ENT, $1, $3, -1 ) }
| IDENT1 SPECIFIER IDENT1    { Term_ent (ENT, $1, $3, -1 ) }
| LPAR expr_term RPAR    { $2 }
| LBRA RBRA    { Term_una ( STAR, Term_ent (NIL, "", "", -1) ) }
| STAR    { Term_una ( OPT, Term_ent (NIL, "", "", -1) ) }
| LBRA expr_term RBRA    { Term_una (STAR, $2) }
| LBRA expr_term COMMA expr_star_lst RBRA    { Term_bin (STAR, $2, $4) }
| LSQB expr_term RSQB    { Term_una (CROSS, $2) }
| LSQB expr_term COMMA expr_cross_lst RSQB    { Term_bin (CROSS, $2, $4) }
| LBIL expr_term RBIL    { Term_una (STROK, $2) }
| LBIL expr_term COMMA expr_strok_lst RBIL    { Term_bin (STROK, $2, $4) }
| expr_term OPT    { Term_una (OPT, $1) }
| expr_term LEFT    { Term_una (LEFT, $1) }
| expr_term RIGHT    { Term_una (RIGHT, $1) }
| expr_term WEDGE expr_term    { Term_bin (WEDGE, $1, $3) }
| expr_term VEE expr_term    { Term_bin (VEE, $1, $3) }
;

expr_star_lst:
  expr_term    { $1 }
| expr_term COMMA expr_star_lst    { Term_bin (STAR, $1, $3) }
;
expr_cross_lst:
  expr_term    { $1 }
| expr_term COMMA expr_cross_lst    { Term_bin (CROSS, $1, $3) }
;
expr_strok_lst:
  expr_term    { $1 }
| expr_term COMMA expr_strok_lst    { Term_bin (STROK, $1, $3) }
;

expr_pat:
  IDENT1    { Pat_ent (ENT, $1, -1) }
| LPAR expr_pat RPAR    { $2 }
| expr_pat STAR    { Pat_una (STAR, $1, -1) }
| expr_pat CROSS    { Pat_una (CROSS, $1, -1) }
| expr_pat STROK    { Pat_una (STROK, $1, -1) }
| expr_pat OPT    { Pat_una (OPT, $1, -1) }
| expr_pat ALT expr_pat    { Pat_bin (ALT, $1, $3, -1) }
| expr_pat WEDGE expr_pat    { Pat_bin (WEDGE, $1, $3, -1) }
| expr_pat VEE expr_pat    { Pat_bin (VEE, $1, $3, -1) }
```

Makefile

```
CC = ocamlc
CFLAGS = -c

YACC = ocamlyacc
LEX = ocamllex

RM = rm
TOUCH = touch

EXEC_SUFFIX = cmo
EXP_SUFFIX = cmi


built_done: lie_type.$(EXEC_SUFFIX) lie_equiv.$(EXEC_SUFFIX) lie_match.$(EXEC_SUFFIX) lie_trans.$(EXEC_SUFFIX)\
            compiler_done lie_main.$(EXEC_SUFFIX)
$(TOUCH) $@
compiler_done: lie_type.$(EXEC_SUFFIX) lie_parser.$(EXP_SUFFIX) lie_parser.$(EXEC_SUFFIX) lie_lexer.$(EXEC_SUFFIX)
$(TOUCH) $@


lie_type.$(EXEC_SUFFIX): lie_type.ml
$(CC) $(CFLAGS) $<
lie_equiv.$(EXEC_SUFFIX): lie_equiv.ml
$(CC) $(CFLAGS) $<
lie_match.$(EXEC_SUFFIX): lie_match.ml
$(CC) $(CFLAGS) $<
lie_trans.$(EXEC_SUFFIX): lie_trans.ml
$(CC) $(CFLAGS) $<
lie_main.$(EXEC_SUFFIX): lie_main.ml
$(CC) $(CFLAGS) $<

lie_parser.$(EXEC_SUFFIX): lie_parser.ml
$(CC) $(CFLAGS) $<
lie_parser.$(EXP_SUFFIX): lie_parser.mli
$(CC) $(CFLAGS) $<
lie_parser.ml: lie_parser.mly
$(YACC) $<
lie_parser.mli: lie_parser.mly
$(YACC) $<

lie_lexer.$(EXEC_SUFFIX): lie_lexer.ml
$(CC) $(CFLAGS) $<
lie_lexer.ml: lie_lexer.mll
$(LEX) $<


.PHONY: clean
clean:
-$(RM) a.out
-$(RM) compiler_done built_done
-$(RM) lie_lexer.ml lie_parser.ml lie_parser.mli
-$(RM) *.cmi
-$(RM) *.cmo
-$(RM) *~
-$(RM) \#*
```

.ocamlinit:

```
#load "lie_type.cmo";;
#load "lie_equiv.cmo";;
#load "lie_match.cmo";;
#load "lie_trans.cmo";;
#load "lie_lexer.cmo";;
#load "lie_parser.cmo";;
#load "lie_main.cmo";;
open List
open Lie_type;;
open Lie_equiv;;
open Lie_match;;
open Lie_trans;;
open Lie_main;;
```