Tüübisüsteemid

Plaan

- Motivatsioon
- Esitlus
- Tüübita süsteemid
- Lihtsalt tüübitud süsteemid
- Alamtüüpimine

Kirjandus

1. B. Pierce „Types and Programming Languages“, MIT, 2002

Motivatsioon

Motivatsioon

- Kergekaalulised formaalsed meetodid
  - Mudeli kontrollijad
  - Tüübisüsteemid

Motivatsioon

- Milleks on tüübid programmeerimiskeeltes?
  - Veatuvastus
  - Abstraktsioon
  - Dokumentatsioon
  - Keeleturvalisus
  - Efektiivsus

Esitlus

- Funktsionaalne keel
  - λ-arvutus koos aritmeetilise ja Booli konstruktsioonitega
  - Call-by-value
- Struktuurne operatsioonsemantika
  - Kasutan $\rightarrow$ $\Rightarrow$ asemel.
- Struktuursed tüübid

Tüübita süsteemid

Tüübita aritmeetliliste avaldiste süntaks

t ::=
- true
- false
- if t then t else t
- 0
- succ t
- pred t
- iszero t

Boole'i avaldised

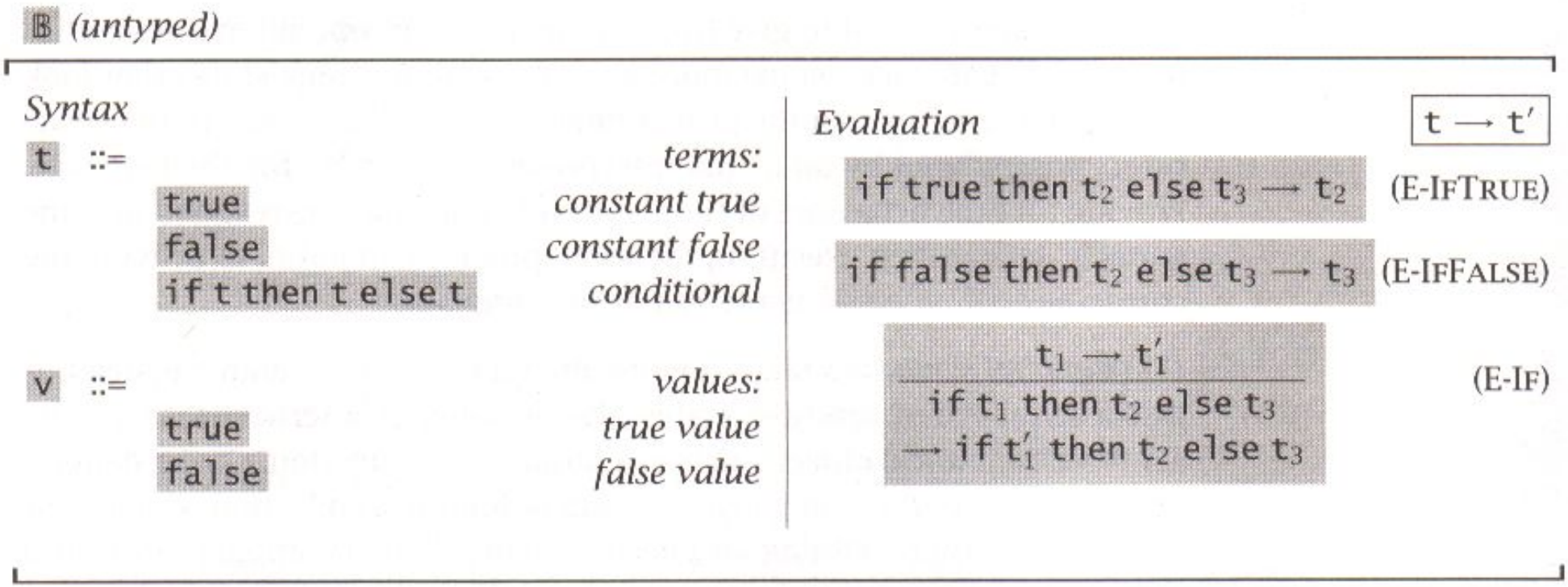


Figure 3-1: Booleans (B)

Aritmeetilised avaldised

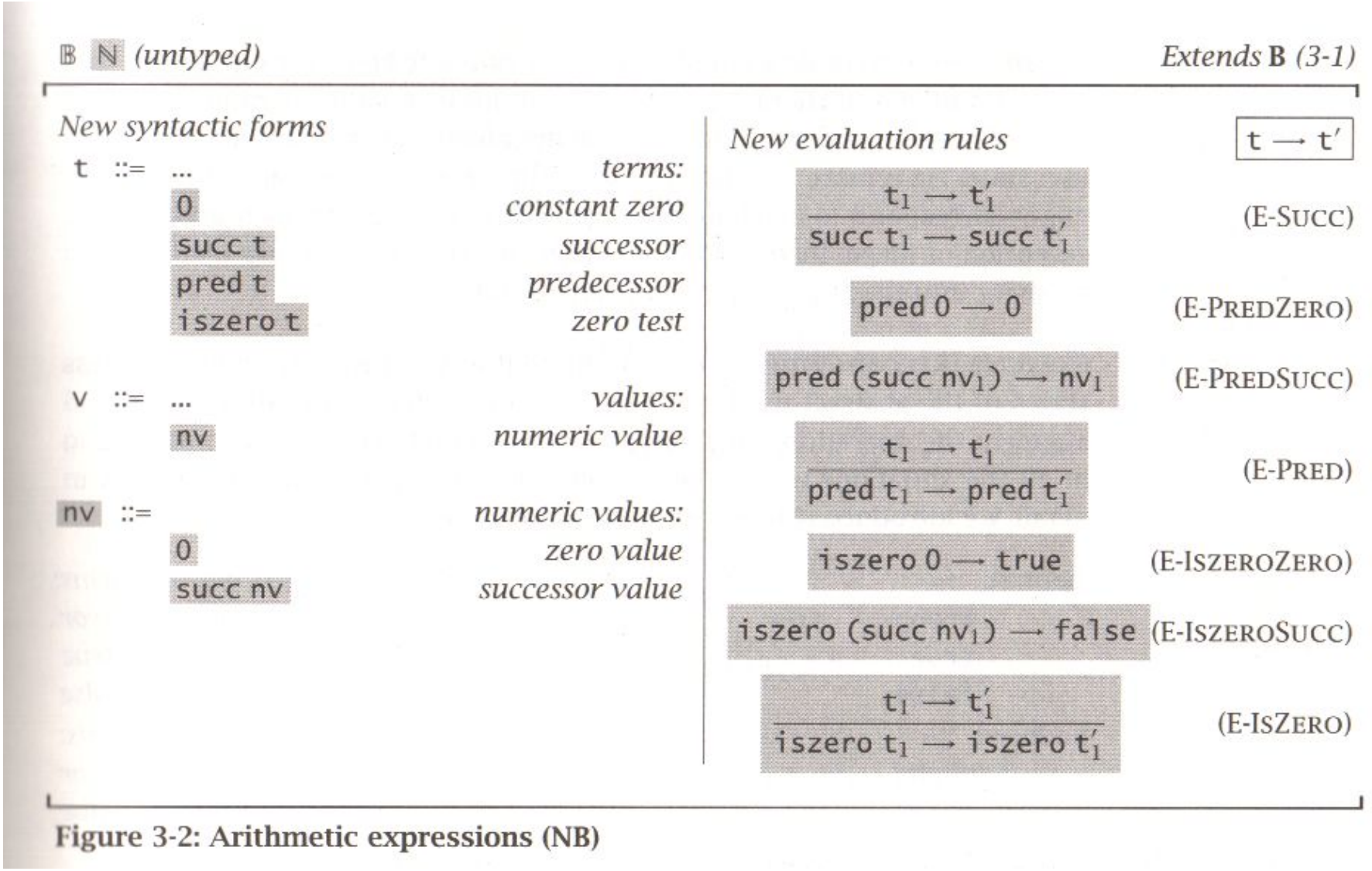


Figure 3-2: Arithmetic expressions (NB)

Normaalkuju

- DEF: Term t on normaalkujul, kui ei leidu t' nii, et $t \rightarrow t'$.
- Teoreem: Iga väärtus on normaalkujul.
  - Kehtib ka teiste keelte kohta, mida me vaatame
- Teoreem: Kui t on normaalkujul, siis t on väärtus.
  - Kehtib hetkel ainult Boole'i avaldiste keeles

Normaalkuju ...

- Teoreem: Kui $t \rightarrow^* u$ ja $t \rightarrow^* u'$, kus u ja u' on normaalkujul, siis $u = u'$.
  - Kehtib, kui keel on ühene; need, mis me vaatame, on
- Teoreem: Iga termi t jaoks leidub normaalkuju t' nii, et $t \rightarrow^* t'$.
- DEF: Term on kinni jäänud („stuck“), kui ta on normaalkujul, aga ei ole väärtus.

λ-arvutus

- Kõik termid on kujul:

  t ::=

  x

  λx.t

  t t

- Näited:

  λx.x

  λxy.y x

  λfxy.f (x y)

λ-arvutus

- DEF: Vabad muutujad FV(t):

$$FV(x)=\{x\}$$
$$FV(\lambda x.t_1)=FV(t_1)\setminus\{x\}$$
$$FV(t_1\ t_2)=FV(t_1)\cup FV(t_2)$$

- DEF: Substitutsioon $[x \to s]\ t$:

$$[x\to s]x=s$$
$$[x\to s]y=y \quad \text{if } y\neq x$$
$$[x\to s](\lambda y.t_1)=\lambda y.[x\to s]t_1 \quad \text{if } y\neq x\wedge y\notin FV(s)$$
$$[x\to s](t_1\ t_2)=([x\to s]t_1)([x\to s]t_2)$$

λ-arvutus

- Näited:
  - $[x \to (\lambda z.z\ w)]\ (\lambda y.x) = \lambda y.\lambda z.z\ w$ (õige)
  - $[x \to y]\ (\lambda x.x) = \lambda x.y$ (vale)
  - $[x \to z]\ (\lambda z.x) = \lambda z.z$ (vale)
  - $[x \to y\ z]\ (\lambda y.x\ y) = ?$
- Meid ei huvita muutujate nimed
  - Valime alati unikaalsed muutujad
  - Nimetame ümber, kui tekivad konfliktid

λ-arvutus

→ *(untyped)*

Syntax

t ::=		*terms:*
	x	*variable*
	λx.t	*abstraction*
	t t	*application*
v ::=		*values:*
	λx.t	*abstraction value*

Evaluation

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-App2)}$$

$$(\lambda x.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

$t \longrightarrow t'$

Figure 5-3: Untyped lambda-calculus (λ)

λ-arvutuse intuitsioon

- Lambdaabstraktsioon on funktsiooni definitsioon
- Aplikatsioon on funktsiooni rakendamine
- Betareduktsioon on avaldise väärtustamine
  - Termi osa, mida saab redutseerida, nimetatakse reedeksiks
- DEF: Termi, milles ei leidu vabu muutujaid, nimetatakse kinniseks.

Näited

- $\underline{(\lambda x.x)\ (y\ z)} \rightarrow [x \rightarrow y\ z]\ x \rightarrow y\ z$
- $\underline{(\lambda x.\ y)\ w} \rightarrow y$
- $\underline{(\lambda xy.y\ x)\ z}\ w \rightarrow \underline{(\lambda y.y\ z)\ w} \rightarrow w\ z$
- $\underline{(\lambda x.x\ x)\ (\lambda x.x\ x)} \rightarrow \underline{(\lambda x.x\ x)\ (\lambda x.x\ x)} \rightarrow \ldots$

Currying

- Kuid me ei saa esitada otseselt λ-arvutuses mitmeargumendiliseid funktsioone, saame teha funktsioone, mis tagastavad tulemusena funktsioonid (*high order functions*):

$\underline{(\lambda xy.\text{add } x\ y)\ 3}\ 4 \rightarrow \underline{(\lambda y.\text{add } 3\ y)\ 4} \rightarrow$

$\underline{\text{add } 3\ 4} \rightarrow 7$

Boole'i funktsioonid

- Boole'i funtksioonid λ-arvutuses:
  - tru = λt. λf. t
  - fls = λt. λf. f
  - test = λl. λm. λn. l m n
  - and = λb. λc b c fls
  - ...

Rekursioon

- Püsipunktikombinaator:

fix = λf. (λx. f (λy. x x y)) (λx. f (λy. x x y))

(on ka lihtsam kombinaator, aga seda saab kasutada ainult *call-by-name* reduktsioonjadaga)

- Näide:

fct = λf. λn. if n = 0 then 1
  else n * (f (n-1))

factorial = fix fct

Lihtsalt tüübitud süsteemid

Tüübid

- Milleks?
  - Me ei taha kinni jäänud terme
- Kuidas?
  - Igal väärtusel on oma kindel tüüp
  - Termil t on tüüp T, kui ta ilmselt redutseerub väärtuseks tüübiga T („ilmselt“ tähendab siin, et me saame leida termi tüübi ilma väärtustamata)
  - Term t on tüübitav, kui leidub tüüp T nii, et see on t tüüp

Booli tüübid

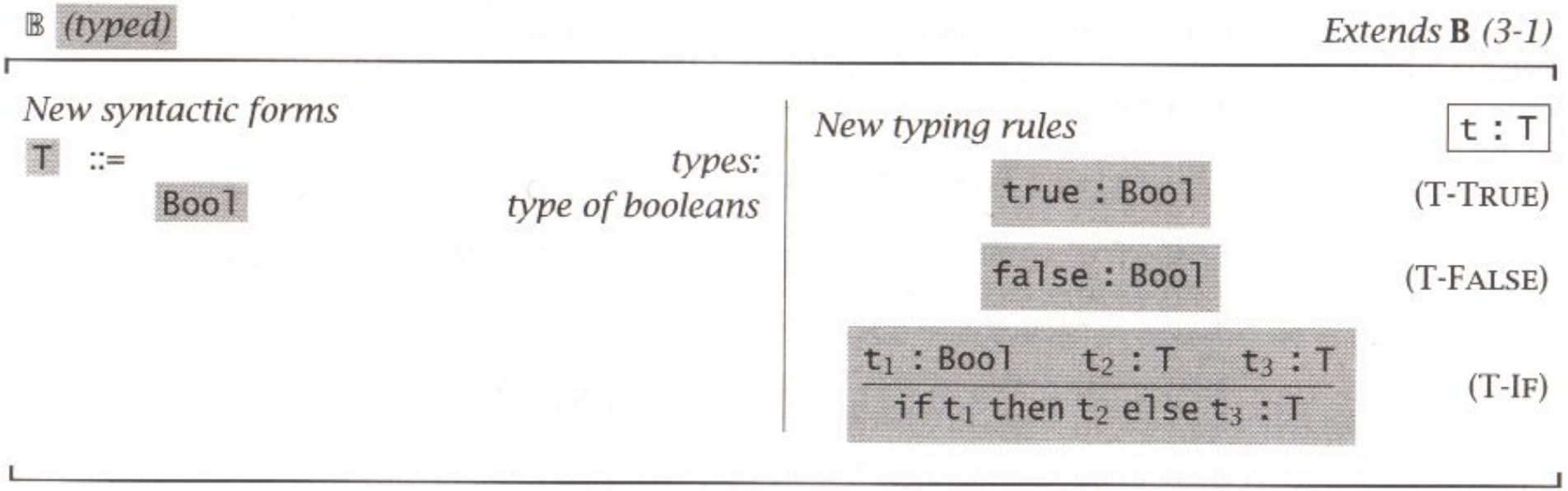


Figure 8-1: Typing rules for booleans (B)

Arimeetilised tüübid

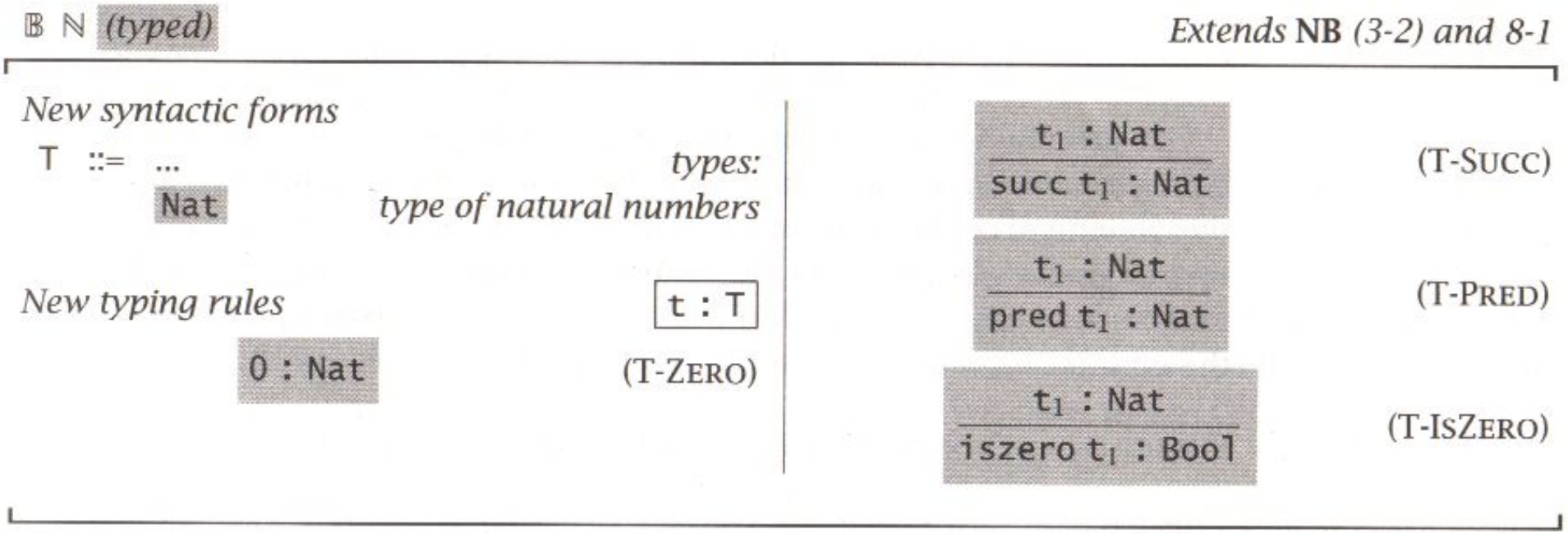


Figure 8-2: Typing rules for numbers (NB)

Tüüpide tööriistakast

- Inversioonilemma
  - Väljastab tüübileidmise algoritmi
- Tüübi unikaalsuse teoreem
  - Tagab, et algoritm leiab õige tüübi
- Edenemine + Säilitamine = Turvalisus
  - Tagab, et tüübitav term ei saa kunagi kinni jääda

Inversioonilemma

1. Kui true : R, siis R = Bool
2. Kui false : R, siis R = Bool
3. Kui if t_1 then t_2 else t_3 : R, siis t_1 : Bool, t_2 : R ja t_3 : R
4. Kui 0 : R siis R = Nat
5. Kui succ t : R või pred t : R, siis R = Nat
6. Kui iszero t : R siis R = Bool ja t : Nat

Tüübi unikaalsuse teoreem

- Igal termil t on mitte rohkem kui üks tüüp.
- Teiste sõnadega, kui term t on tüübitav, siis see tüüp on unikaalne

Edenemine ja säilitamine

- Edenemise teoreem
  - Tüübitav term ei ole kinni jäänud.
  - S.t. kas term on väärtus või leidub üleminek mingi väärtustamise reegli järgi
- Säilitamise teoreem
  - Kui tüübitav term teeb ülemineku, siis tulemuseks on ka tüübitav term.
  - NB-keeles kehtib isegi tugevam teoreem - tulemuseks on term sama tüübiga (hilisemate keelte kohta see enam ei kehti)

Lihtsalt tüübitud λ-arvutus

- Funktsioonide tüübid
  - $T \to T$
  - $\lambda x{:}T.\ t$
- Tüübitavus
  - Ilmutatud
  - Ilmutamata

Lihtsalt tüübitud λ-arvutus

- Tüübikontekst
  - $\Gamma = \{x_1 : T_1, x_2 : T_2, \ldots\}$
  - $\Gamma \supset t : T$
  - Muutuja tüüp on võetud kontekstist
  - Abstraktsioon täiendab konteksti oma argumendi tüübiga

Lihtsalt tüübitud λ-arvutus

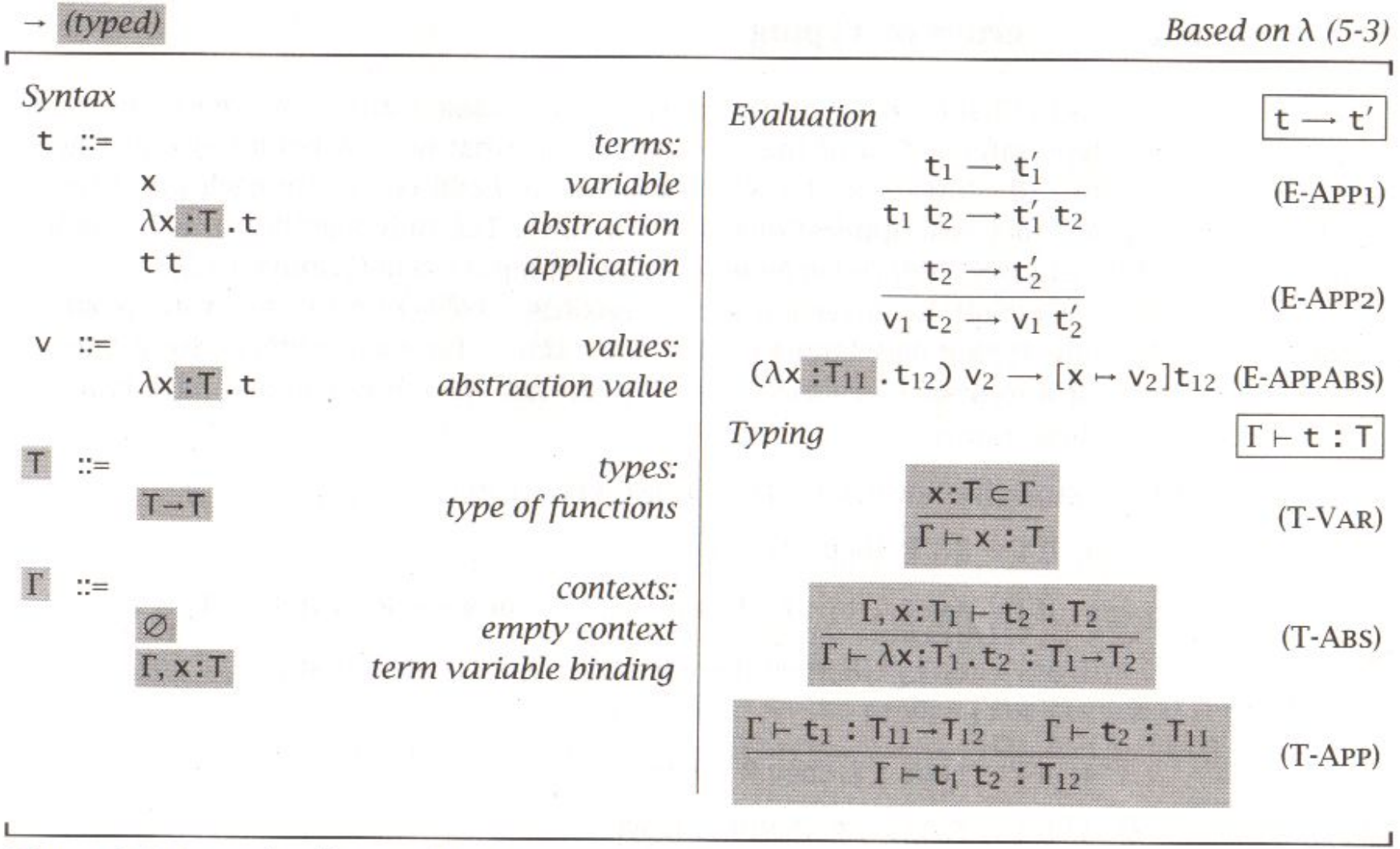


Figure 9-1: Pure simply typed lambda-calculus (λ_)

λ-arvutuse tööriistakast

- Inversioonilemma
  1. Kui $\Gamma \supset x : R$, siis $x : R \in \Gamma$
  2. Kui $\lambda x{:}T_1.\ t_2 : R$, siis $R = T_1 \to R_2$, mingi R_2 korda ja $\Gamma, x : T_1 \supset t_2 : R_2$
  3. Kui $\Gamma \supset t_1\ t_2 : R$, siis leidub mingi T_{11} nii, et $\Gamma \supset t_1 : T_{11} \to R$ ja $\Gamma \supset t_2 : T_1$

λ-arvutuse tööriistakast

- Unikaalsuse teoreem
  - Antud kontekstis leidub termil ülimalt üks tüüp (kui kõik termi vabad muutujad on esitatud kontekstis).
- Edenemise teoreem
  - Olgu t kinnine tüübitav term (leidub T nii, et $\supset$ t : T). Siis kas t on väärtus või leidub t' nii, et t $\rightarrow$ t'.
- Säilitamise teoreem
  - Kui $\Gamma \supset$ t : T ja t $\rightarrow$ t', siis $\Gamma \supset$ t' : T

Lihtsalt tüübitud λ-arvutus

- Tüüpide kustutamine
  - Jätame kõik tüübiannotatsioonid ära
  - Saame, et meie tüübitav termid on tüübitamata termide alamhulk ja termide reduktsioon on säilitatud
- Curry v/s Church stiilid
  - Curry stiil: t võib omada tüüpi, siis see on tüübitav
  - Church stiil: me ei vaata terme, mis ei oma tüüpi

Laiendused

Lisatüübid

- Baastüübid (interpreteerimata tüübid)
  - Nendega pole seotud ühtegi redutseerimisreeglit
  - Saame kasutada funktsioonides
- Ühiktüüp
  - `()` Haskellis
  - `void` Javas

Baastüübid

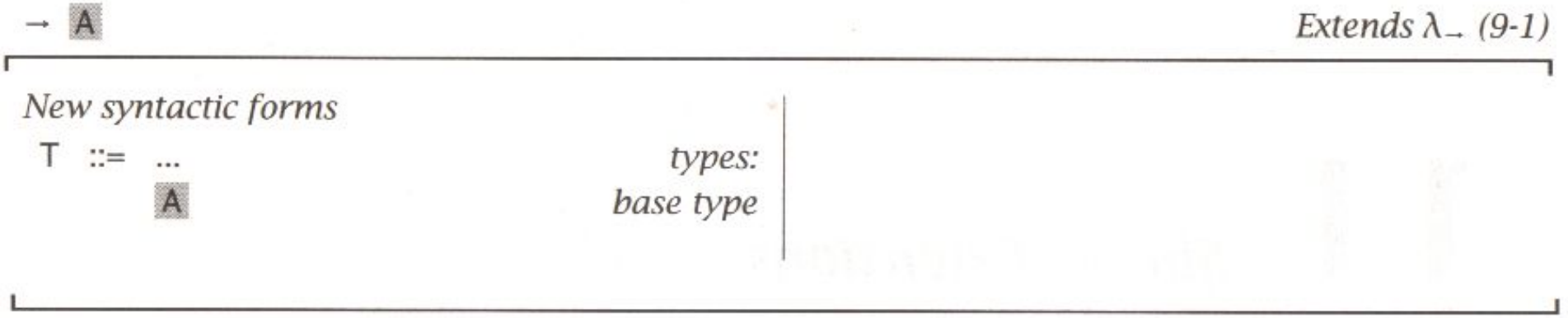


Figure 11-1: Uninterpreted base types

Tuletatud kuju

- Kompositsioon
  - $t_1 ; t_2 = (\lambda x : \text{Unit}.\ t_2)\ t_1$, kus $x \notin FV(t_2)$
  - Saab defineerida ka täisreeglitega
  - Tuletatud kuju nimetatakse tihti ka süntaktiliseks suhkruks
- Metamärgid (Wildcards)
  - Kui meil pole vaja λ-abstraktsiooni argumenti, siis kirjutame $(\lambda\_ : S.\ t)$.

Ühiktüüp

→ Unit — *Extends* $\lambda_{\to}$ *(9-1)*

New syntactic forms

t	::=	...	*terms:*
		unit	*constant* unit
v	::=	...	*values:*
		unit	*constant unit*
T	::=	...	*types:*
		Unit	*unit type*

New typing rules

$\Gamma \vdash t : T$

$$\Gamma \vdash \texttt{unit} : \texttt{Unit}$$ (T-UNIT)

New derived forms

$$t_1;t_2 \stackrel{\text{def}}{=} (\lambda x{:}\texttt{Unit}.t_2)\ t_1 \quad \text{where } x \notin FV(t_2)$$

Figure 11-2: Unit type

Omistus

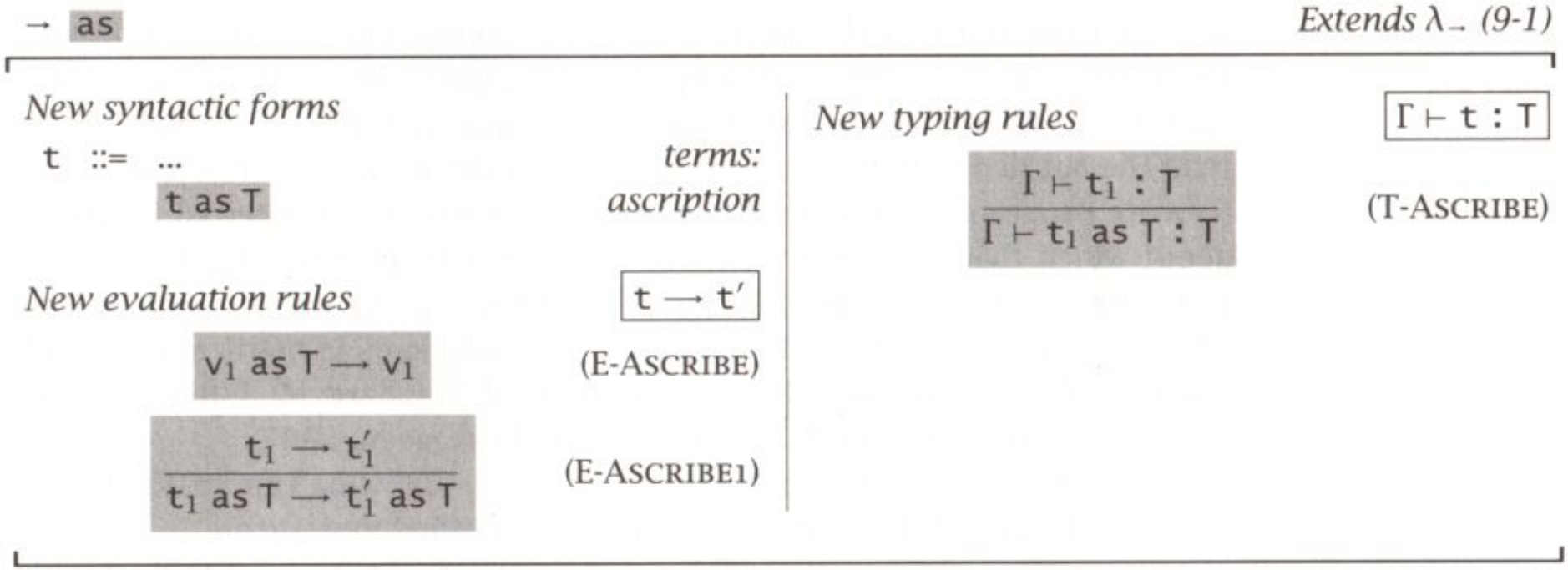


Figure 11-3: Ascription

Omistus

- Dokumentatsioon
  - Aitab ka programmeerijal mõista
- Tüüpide printimine
- Abstraktsioon
  - Tüübi määramine (eriti tähtis alamtüüpimisel)
  - Nt. Javas casting

Let-sidumine

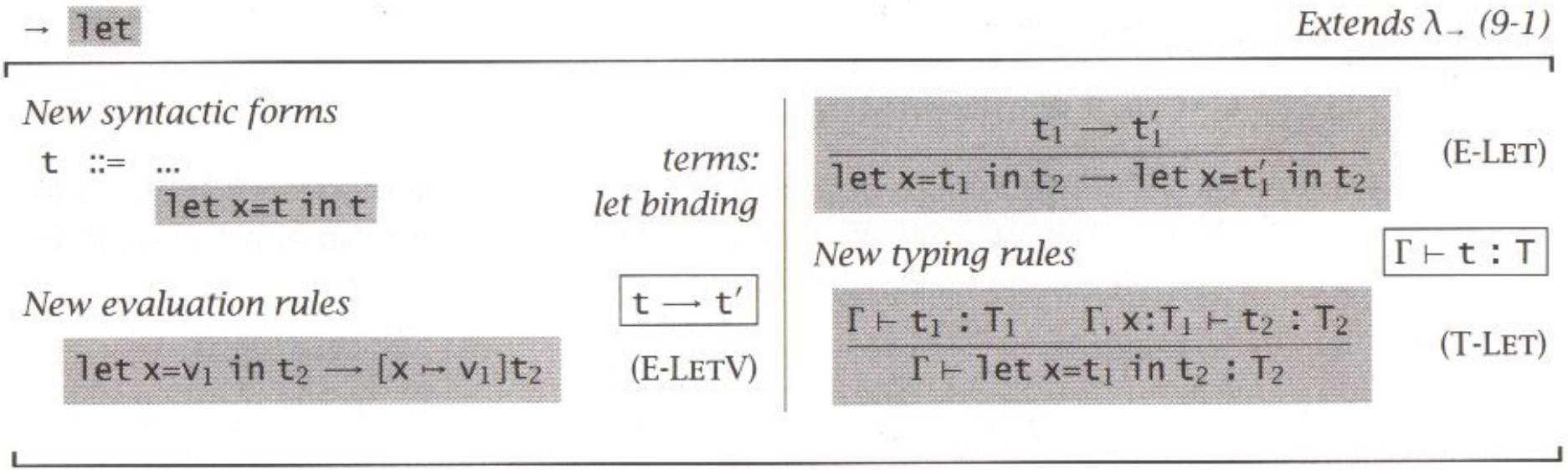


Figure 11-4: Let binding

Let-sidumist saab defineerida ka tuletatud kujuna, kuid siis peame leidma tüübi tüübikontrollijalt

Paarid

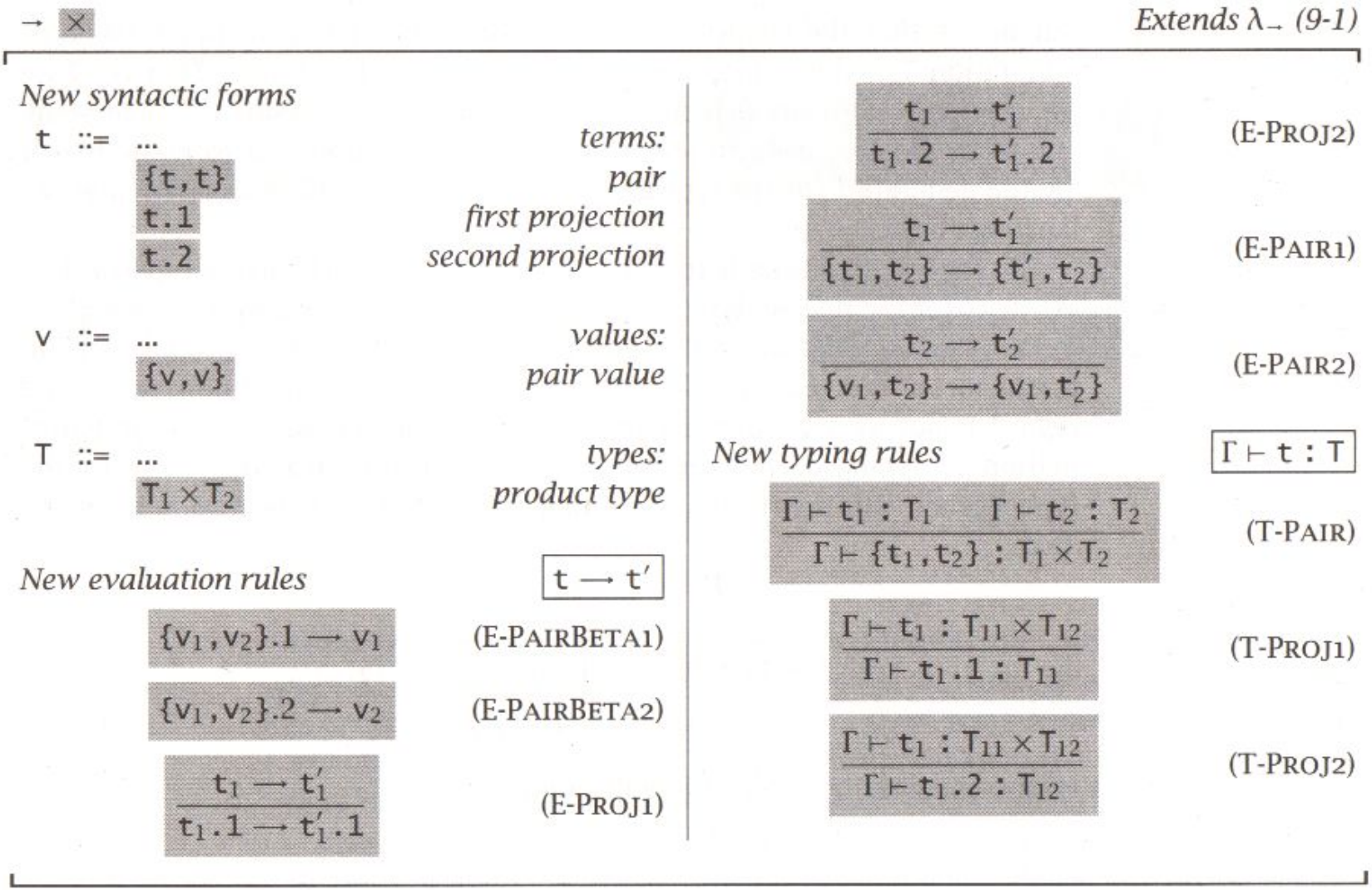


Figure 11-5: Pairs

Korteežid (Tuples)

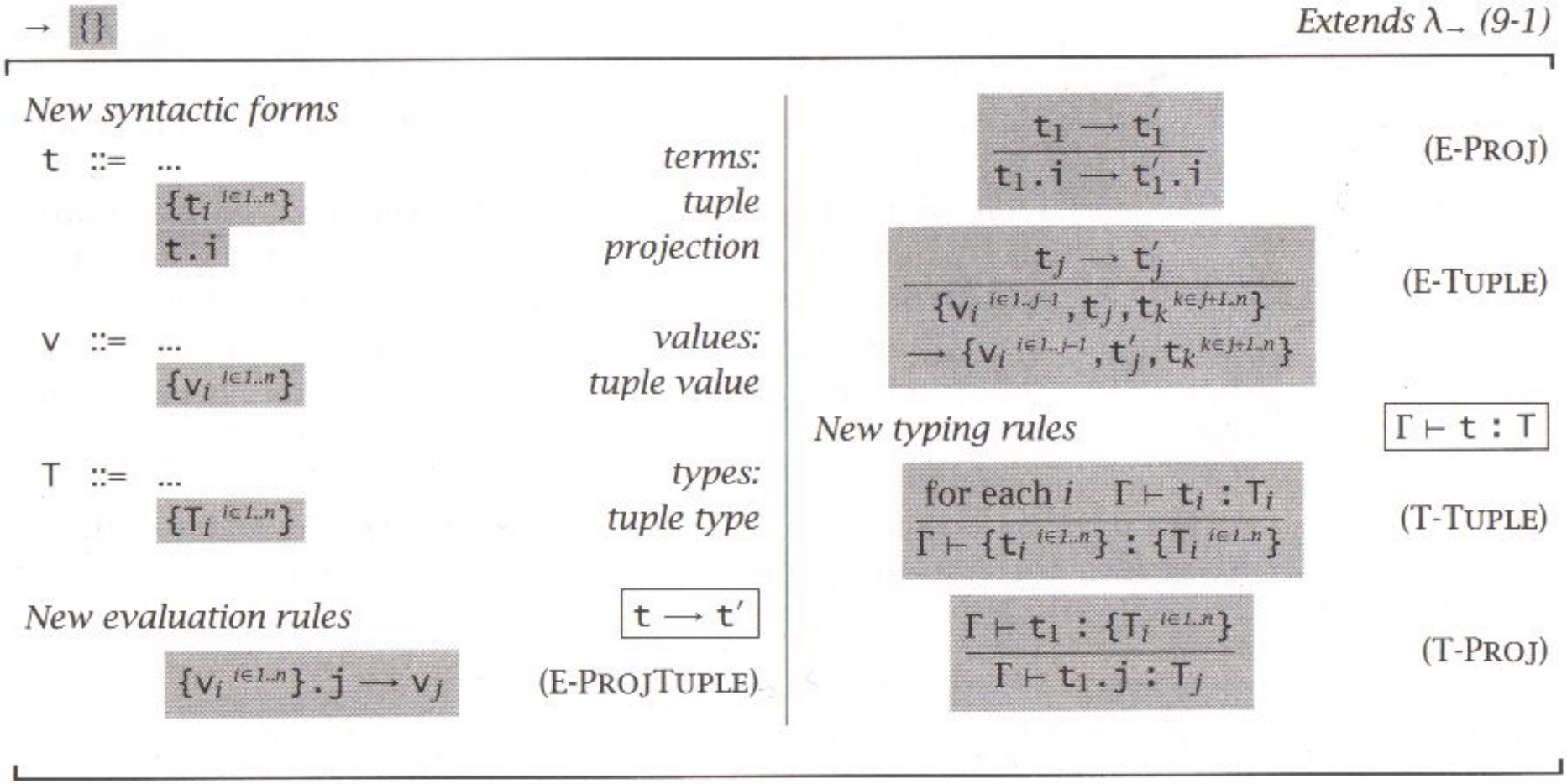


Figure 11-6: Tuples

Kirjed

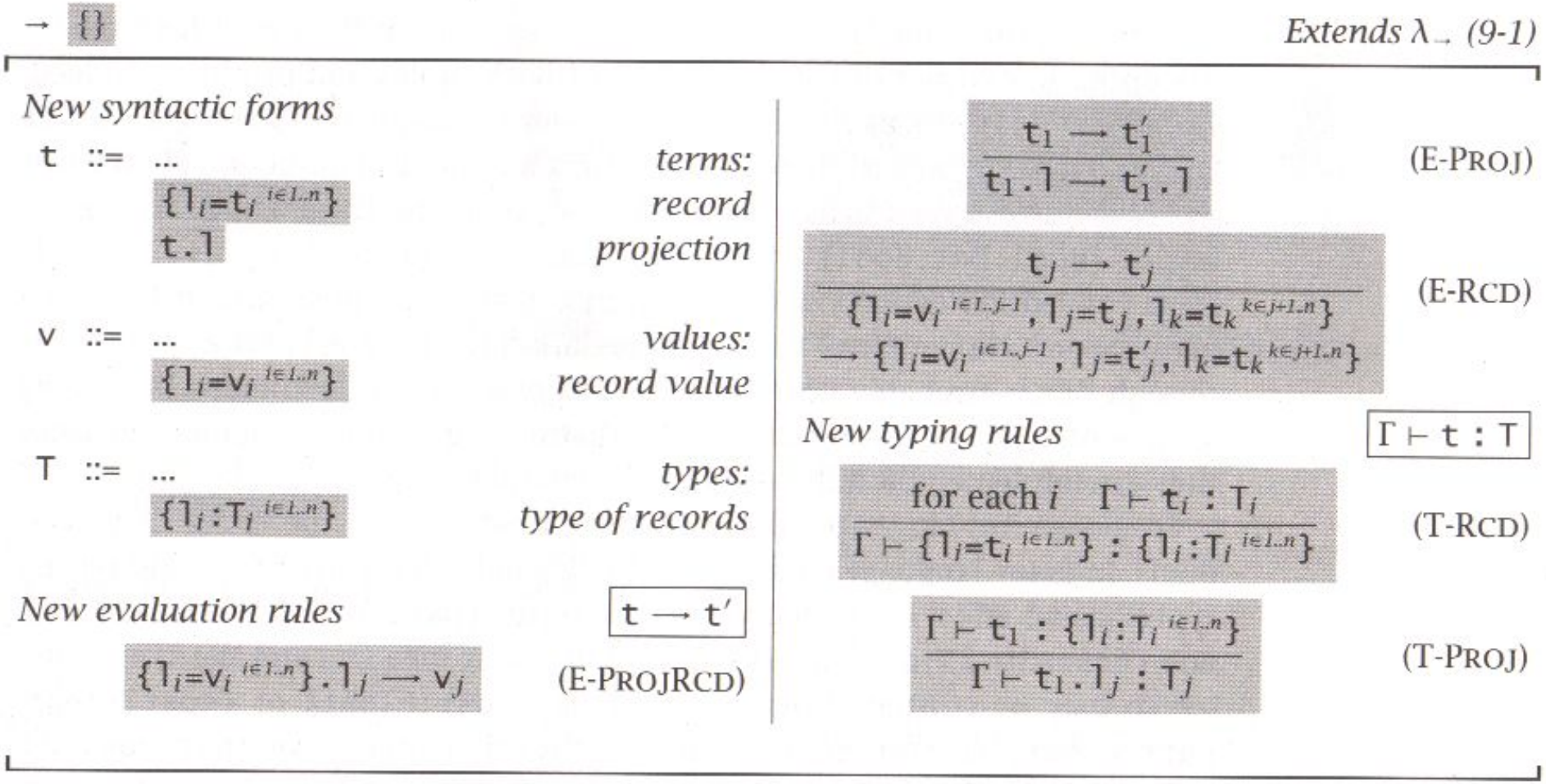


Figure 11-7: Records

Summad

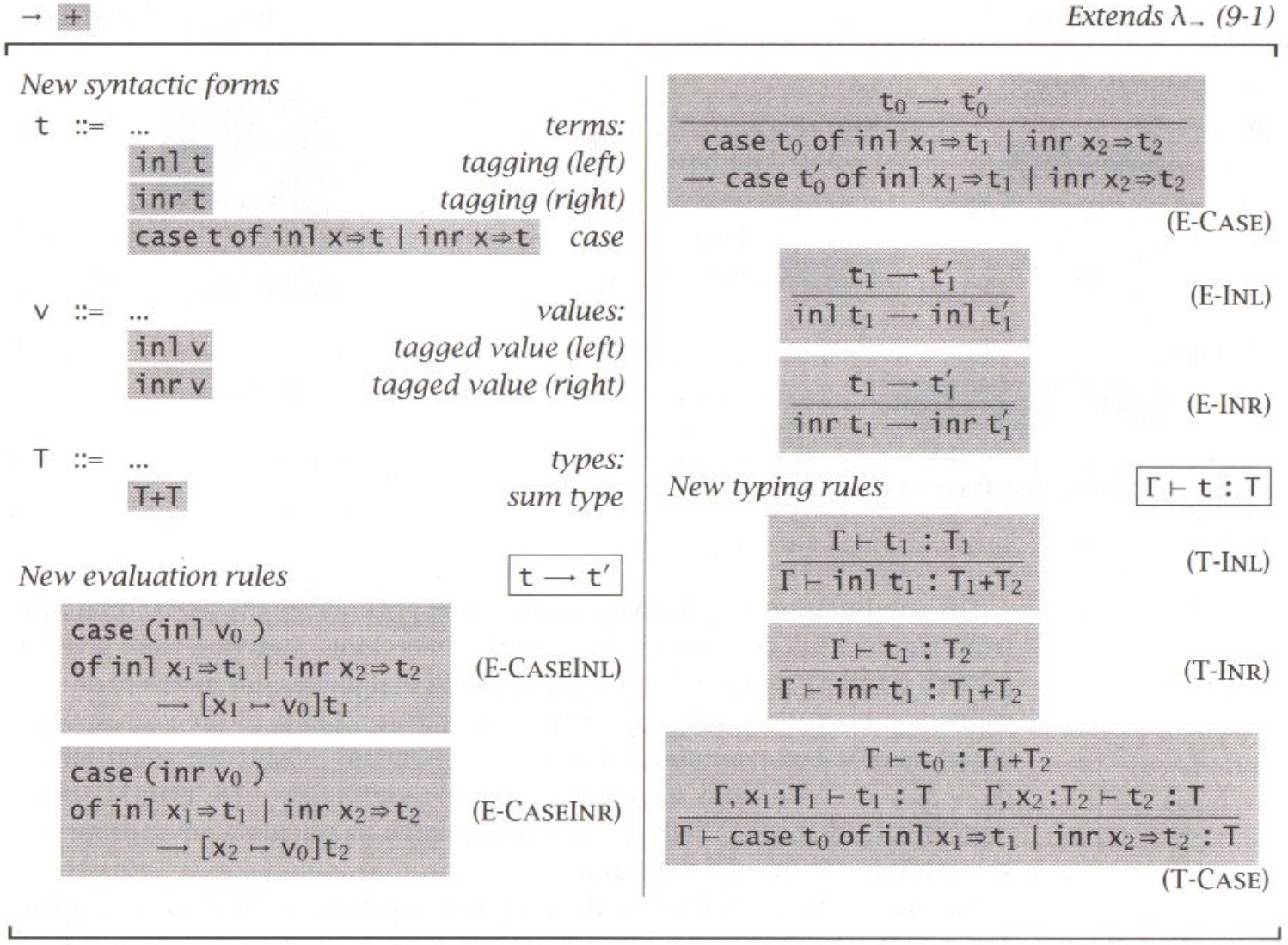


Figure 11-9: Sums

Summad ja tüübiunikaalsus

- Mis on termi inl t tüüp, kui t : T?
  - T + A?
  - T + B?
  - Kui T = Nat, siis võiks olla Nat+Nat, aga võiks ka Nat+Bool

Summad ja tüübiunikaalsus

- Lahendusviisid

  1)Tüübikontrollija võib kuidagi proovida arvata tüübi (täpsemalt: ta võib jätta tüübi määramata, kuni see selgub)

  2)Võime muuta keelt nii, et see võimaldaks meil esitada kõik võimalikud tüübid samasuguselt (seda kasutame, kui hakkame alamtüüpimist vaatama)

  3)Võime nõuda programmeerijalt, et ta paneks kirja tüübi ilmutatud kujul

Summad tüübiomistusega

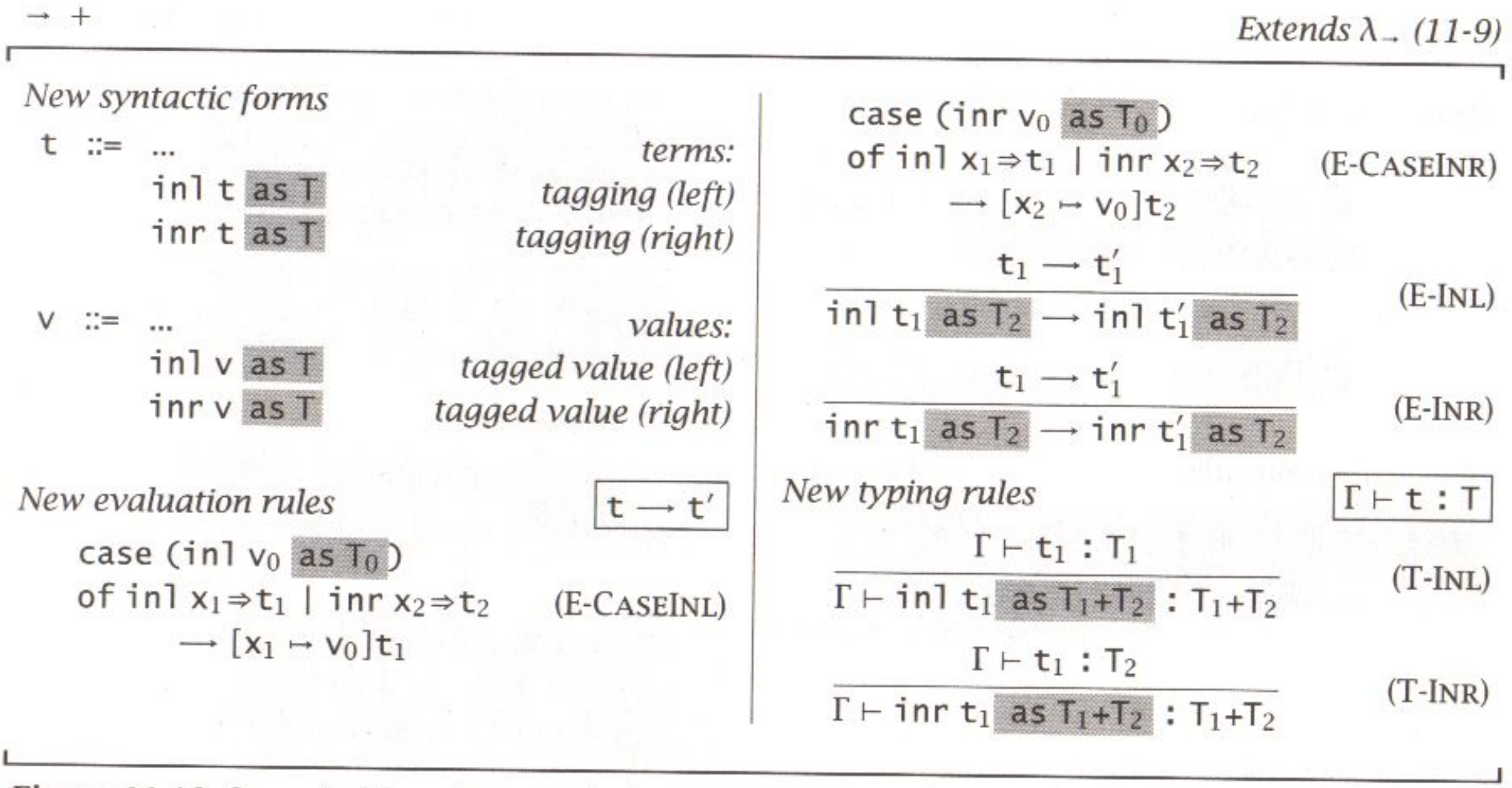


Figure 11-10: Sums (with unique typing)

Variandid

→ <>

Extends $\lambda_{\to}$ *(9-1)*

New syntactic forms

t	::=	...	*terms:*
		$\langle l{=}t \rangle$ as T	*tagging*
		case t of $\langle l_i{=}x_i \rangle \Rightarrow t_i \ ^{i \in 1..n}$	*case*
T	::=	...	*types:*
		$\langle l_i : T_i \ ^{i \in 1..n} \rangle$	*type of variants*

New evaluation rules $t \longrightarrow t'$

$$\text{case } (\langle l_j{=}v_j \rangle \text{ as T}) \text{ of } \langle l_i{=}x_i \rangle \Rightarrow t_i \ ^{i \in 1..n} \longrightarrow [x_j \mapsto v_j]t_j \qquad \text{(E-CaseVariant)}$$

$$\frac{t_0 \longrightarrow t_0'}{\text{case } t_0 \text{ of } \langle l_i{=}x_i \rangle \Rightarrow t_i \ ^{i \in 1..n} \longrightarrow \text{case } t_0' \text{ of } \langle l_i{=}x_i \rangle \Rightarrow t_i \ ^{i \in 1..n}} \qquad \text{(E-Case)}$$

$$\frac{t_i \longrightarrow t_i'}{\langle l_i{=}t_i \rangle \text{ as T} \longrightarrow \langle l_i{=}t_i' \rangle \text{ as T}} \qquad \text{(E-Variant)}$$

New typing rules $\Gamma \vdash t : T$

$$\frac{\Gamma \vdash t_j : T_j}{\Gamma \vdash \langle l_j{=}t_j \rangle \text{ as } \langle l_i : T_i \ ^{i \in 1..n} \rangle : \langle l_i : T_i \ ^{i \in 1..n} \rangle} \qquad \text{(T-Variant)}$$

$$\frac{\Gamma \vdash t_0 : \langle l_i : T_i \ ^{i \in 1..n} \rangle \quad \text{for each } i \quad \Gamma, x_i : T_i \vdash t_i : T}{\Gamma \vdash \text{case } t_0 \text{ of } \langle l_i{=}x_i \rangle \Rightarrow t_i \ ^{i \in 1..n} : T} \qquad \text{(T-Case)}$$

Figure 11-11: Variants

Variantide tüübid

- Valikulised tüübid (optional)
  - `Maybe` in Haskell
- Loetelu tüübid
  - `enum` in Java 1.5
  - Tühjade konstruktoritega Haskellis
- Ühe välja variant
  - Pakkimistüübid Haskellis
- Varianti võib vaadata kui eraldatud ühend (disjoint union)

Rekursioon

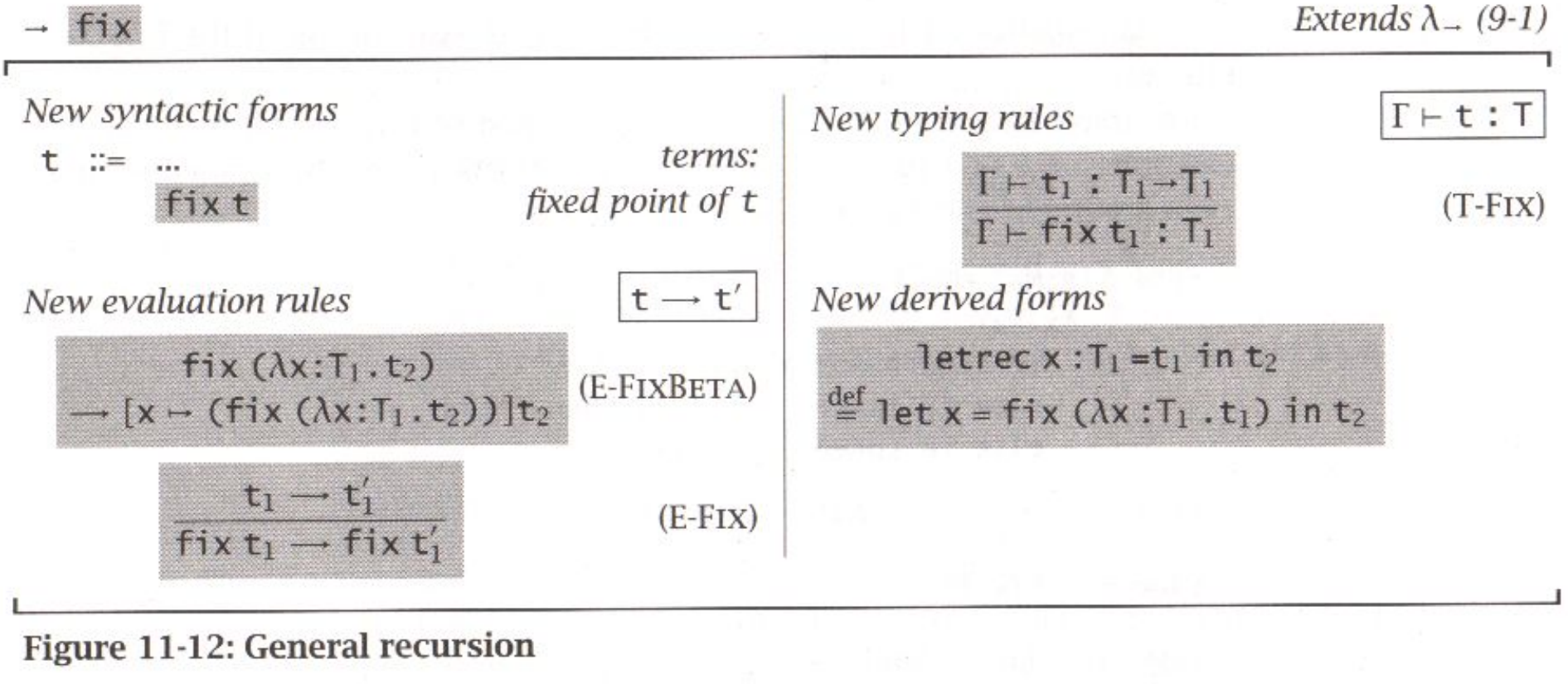


Figure 11-12: General recursion

Tavaline püsipunktikombinaator lambdaarvutuses ei ole tüübitav lihtsalt tüübitavas lambdaarvutuses

Normaliseerimine

- DEF: Term t on normaliseeritav, kui selle võib redutseerida normaalkujuks lõpliku arvu üleminekutega.
- Lihtsalt tüübitud lambdaarvutuses on kõik kinnised tüübitavad termid normaliseeritavad
  - Kehtib niikaua, kui me ei too üldist rekursiooni ja rekursiivseid tüüpe

Viited (references)

- Mida me tahame saada?
  - r = ref 5; ⇒ r : Ref Nat
  - !r; ⇒ 5 : Nat
  - r := 7; ⇒ unit : Unit
  - !r; ⇒ 7 : Nat
  - s := 10; !s ⇒ 10 : Nat

Viited ...

- Jagatud olek
  - Viited võimaldavad suhelda programmi osade vahel
  - Viited võimaldavad teha primitiivseid objekte
  - Viited võimaldavad viidata ka liittüüpidele, nii saab teha nt. massiive
- Prügi vedamine (Garbage collection)
  - Keel ilmutatud vabastamisega ei saa olla turvaline

Viidete esitlus

- Tüüpimine (kontekstis)
  - t : T ⇒ ref t : Ref T
  - t : Ref T ⇒ !t : T
  - rt : Ref T & t : T ⇒ rt := t : Unit
- Väärtustamine
  - Millised on Ref T väärtused?
  - Toome sisse varu (store) $\mathscr{L}$, ref t väärtus on asukoht varus l
  - Iga reduktsioon võib muuta varu

Viidete esitlus ...

- Varu tüüpimine
  - Kuidas tüüpida varu asukohta?
  - Toome sisse varu tüüpimiskonteksti Σ

$$\frac{\Sigma(l) = T_1}{\Gamma | \Sigma \supset l : Ref\ T_1} \qquad \frac{\Gamma | \Sigma \supset t_1 : T_1}{\Gamma | \Sigma \supset ref\ t_1 : Ref\ T_1}$$

$$\frac{\Gamma | \Sigma \supset t_1 : Ref\ T_1}{\Gamma | \Sigma \supset !\, t_1 : T_1} \qquad \frac{\Gamma | \Sigma \supset t_1 : Ref\ T_1 \qquad \Gamma | \Sigma \supset t_2 : T_1}{\Gamma | \Sigma \supset t_1 := t_2 : Unit}$$

Viited (süntaks)

t ::=		*terms:*
	x	*variable*
	λx:T.t	*abstraction*
	t t	*application*
	unit	*constant* unit
	ref t	*reference creation*
	!t	*dereference*
	t:=t	*assignment*
	l	*store location*
v ::=		*values:*
	λx:T.t	*abstraction value*
	unit	*constant unit*
	l	*store location*
T ::=		*types:*
	T→T	*type of functions*
	Unit	*unit type*
	Ref T	*type of reference cells*

Γ ::=		*contexts:*
	∅	*empty context*
	Γ, x:T	*term variable binding*
μ ::=		*stores:*
	∅	*empty store*
	μ, *l* = v	*location binding*
Σ ::=		*store typings:*
	∅	*empty store typing*
	Σ, *l*:T	*location typing*

Viited (väärtustamine)

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1\ t_2 \mid \mu \longrightarrow t_1'\ t_2 \mid \mu'} \quad \text{(E-App1)}$$

$$\frac{t_2 \mid \mu \longrightarrow t_2' \mid \mu'}{v_1\ t_2 \mid \mu \longrightarrow v_1\ t_2' \mid \mu'} \quad \text{(E-App2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \mid \mu \longrightarrow [x \mapsto v_2]t_{12} \mid \mu \quad \text{(E-AppAbs)}$$

$$\frac{l \notin dom(\mu)}{\texttt{ref}\ v_1 \mid \mu \longrightarrow l \mid (\mu, l \mapsto v_1)} \quad \text{(E-RefV)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\texttt{ref}\ t_1 \mid \mu \longrightarrow \texttt{ref}\ t_1' \mid \mu'} \quad \text{(E-Ref)}$$

$$\frac{\mu(l) = v}{!l \mid \mu \longrightarrow v \mid \mu} \quad \text{(E-DerefLoc)}$$

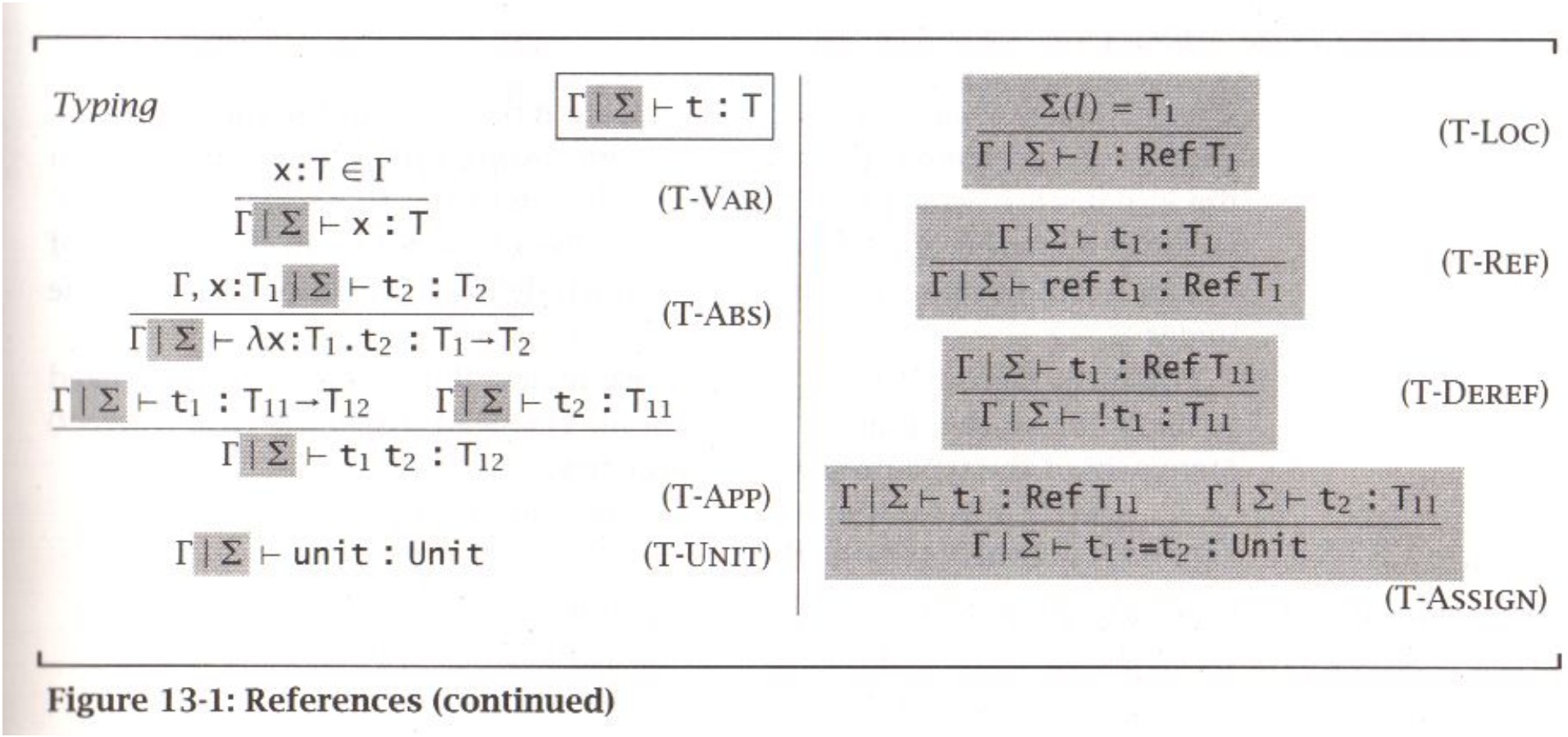
$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{!t_1 \mid \mu \longrightarrow !t_1' \mid \mu'} \quad \text{(E-Deref)}$$

$$l{:=}v_2 \mid \mu \longrightarrow \texttt{unit} \mid [l \mapsto v_2]\mu \quad \text{(E-Assign)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1{:=}t_2 \mid \mu \longrightarrow t_1'{:=}t_2 \mid \mu'} \quad \text{(E-Assign1)}$$

$$\frac{t_2 \mid \mu \longrightarrow t_2' \mid \mu'}{v_1{:=}t_2 \mid \mu \longrightarrow v_1{:=}t_2' \mid \mu'} \quad \text{(E-Assign2)}$$

Viited (tüüpimine)



Typing $\boxed{\Gamma \mid \Sigma \vdash t : T}$

$$\frac{x{:}T \in \Gamma}{\Gamma \mid \Sigma \vdash x : T} \quad \text{(T-VAR)}$$

$$\frac{\Gamma, x{:}T_1 \mid \Sigma \vdash t_2 : T_2}{\Gamma \mid \Sigma \vdash \lambda x{:}T_1.t_2 : T_1 \to T_2} \quad \text{(T-ABS)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \mid \Sigma \vdash t_2 : T_{11}}{\Gamma \mid \Sigma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-APP)}$$

$$\Gamma \mid \Sigma \vdash \texttt{unit} : \texttt{Unit} \quad \text{(T-UNIT)}$$

$$\frac{\Sigma(l) = T_1}{\Gamma \mid \Sigma \vdash l : \texttt{Ref}\ T_1} \quad \text{(T-LOC)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : T_1}{\Gamma \mid \Sigma \vdash \texttt{ref}\ t_1 : \texttt{Ref}\ T_1} \quad \text{(T-REF)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \texttt{Ref}\ T_{11}}{\Gamma \mid \Sigma \vdash !t_1 : T_{11}} \quad \text{(T-DEREF)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \texttt{Ref}\ T_{11} \qquad \Gamma \mid \Sigma \vdash t_2 : T_{11}}{\Gamma \mid \Sigma \vdash t_1 := t_2 : \texttt{Unit}} \quad \text{(T-ASSIGN)}$$

Figure 13-1: References (continued)

Turvalisus

- DEF: Varu on tüübitav (kirjutatakse $\Gamma \mid \Sigma \supset \mu$), kui $\mathrm{dom}(\mu) = \mathrm{dom}(\Sigma)$ ja $\forall l \in \mathrm{dom}(\mu)\ (\Gamma \mid \Sigma \supset \mu(l) : \Sigma(l))$
- Säilitamise teoreem
  - Kui $\Gamma \mid \Sigma \supset t : T$, $\Gamma \mid \Sigma \supset \mu$, $t \mid \mu \rightarrow t' \mid \mu$, siis leidub Σ', $\Sigma \subseteq \Sigma'$ nii, et $\Gamma \mid \Sigma' \supset t' : T$ ja $\Gamma \mid \Sigma' \supset \mu'$
  - Teiste sõnadega, kui meil on tüübitavad term ja varu, siis pärast redutseerimist on nad jätkuvalt tüübitavad

Turvalisus ...

- Edenemise teoreem
  - Olgu t kinnine tüübitav term ($\varnothing \mid \Sigma \supset t : T$), siis kas t on väärtus või iga tüübitava varu μ kohta leidub term t' ja varu μ' nii, et
    $t \mid \mu \rightarrow t' \mid \mu$.
  - Teiste sõnadega, kas term on väärtus või seda saab redutseerida edasi

Erandid

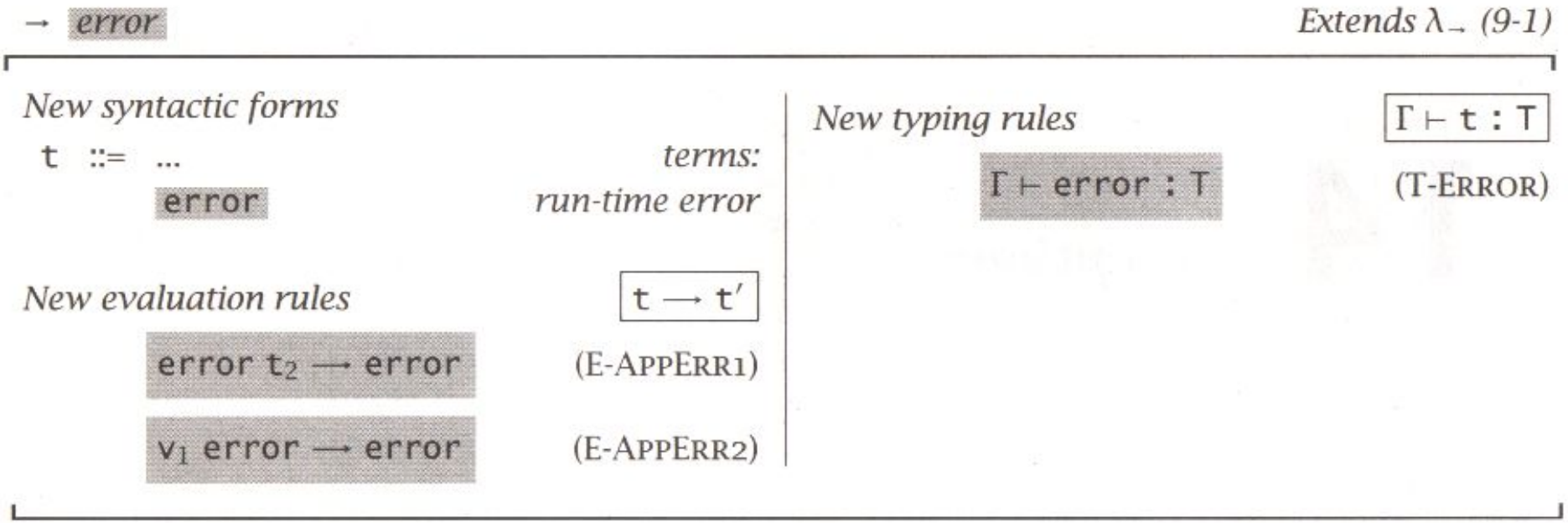


Figure 14-1: Errors

Erandid ...

- Determineeritud
  - $(\lambda x{:}Nat.0)\ error \rightarrow error$
  - $(fix\ (\lambda x{:}Nat.x))\ error \rightarrow^{*} \ldots$
- Tüüpimine
  - Iga tüüp sobib **error** termile
- Edenemise teoreem
  - Term on kas väärtus, seda võib redutseerida või on see **error**

Erandite käsitlus

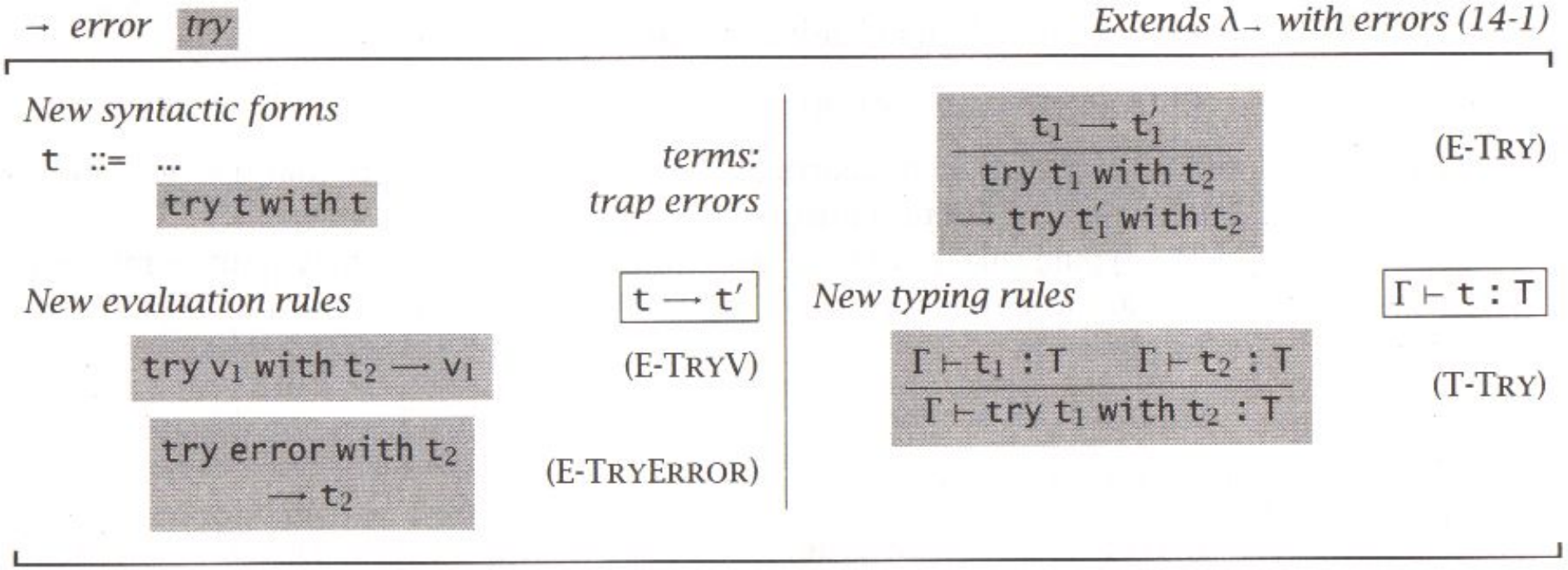


Figure 14-2: Error handling

Erandid väärtustega

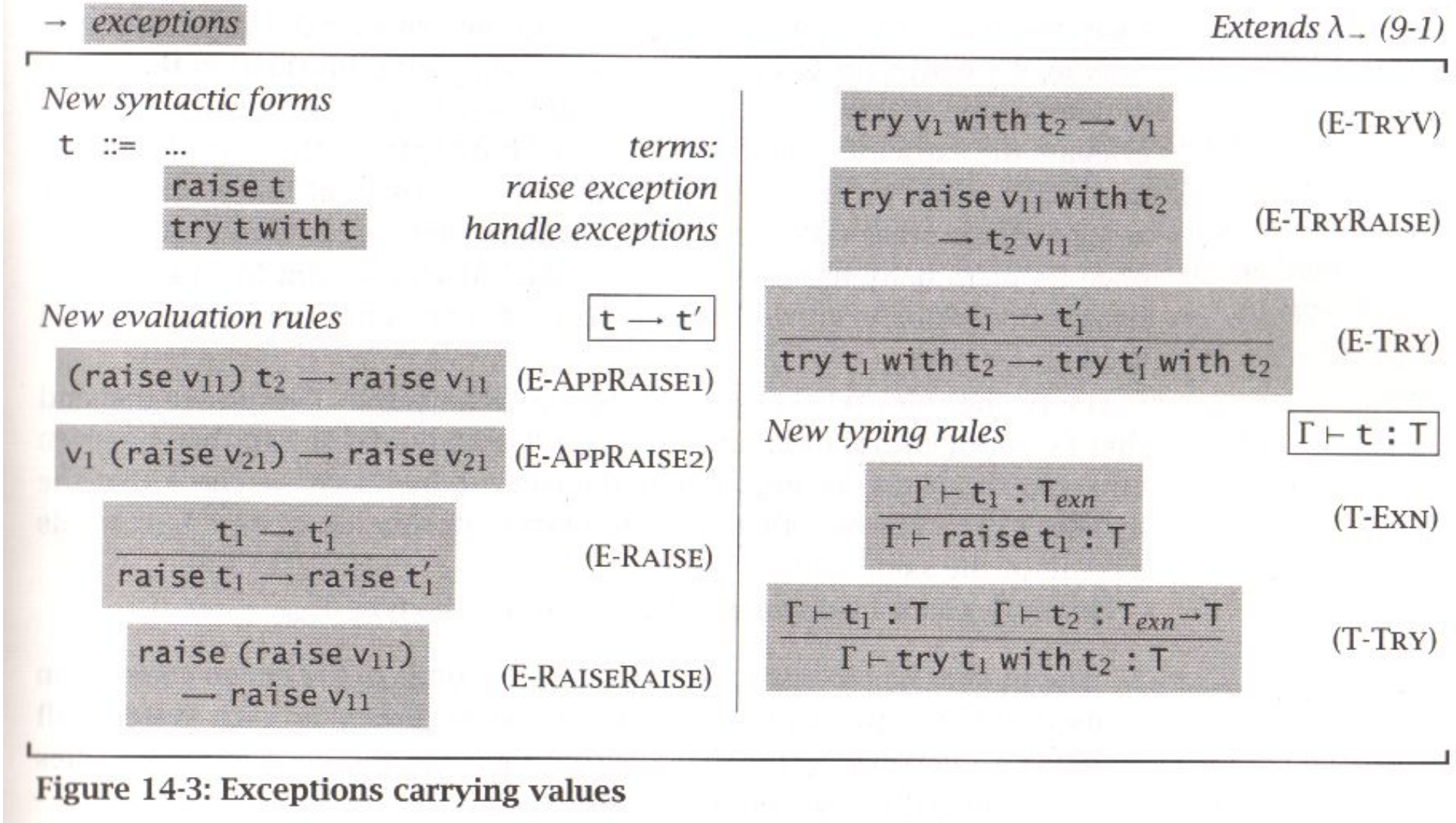


Figure 14-3: Exceptions carrying values

Erandid väärtustega ...

- Mis võib olla T_{exn}?
  - Nat – veakoodid
  - String – veateated
  - Variant – info, mida on vaja
  - Laiendatav variant – nagu MLis
  - Klassid – nagu Javas
    - Lisaks saame ka loomuliku järjestuse

Alamtüüpimine

Motivatsioon

- (λr:{x:Nat}. r.x) {x=0, y=1}
  - Ei ole tüübitav
  - Funktsioon nõuab ainult välja x
    - Saame seda tuletada ainult vaadates funktsiooni argumendi tüüpi
  - On alati ohutu anda {x : Nat} asemel {x : Nat, y : Nat}

Alamtüüpimise relatsioon

- S on T alamtüüp (S <: T), kui igas kontekstis, kus oodatakse T on võimalik kasutada S
  - Üks intuitsioon sellele on, et alamtüüp on alamhulk ülemtüübi väärtustest
- Relatsioon peab olema refleksiivne ja transitiivne
- Subsumtsioon (Subsumption):

$$\frac{\Gamma \supset t : S \quad S <: T}{\Gamma \supset t : T}$$

Kirjete alamtüüpimine

- Kirjetüüp S on kirjetüübi T alamtüüp, kui:
  - T on S väljad teisel järjestusel
  - T on S, kus on lõpust ära jäetud mõni arv väljadest
  - Iga S välja tüüp on sama T välja alamtüüp
- Seega ümberjärjestatud kirjed on ekvivalentsete tüüpidega

Funktsioonide alamtüüpimine

- Mis tingimused peavad olema täidetud, et $S_1 \to S_2 <: T_1 \to T_2$?
  - Kuna funktsiooni kasutakse mingis tundmatus kontekstis, siis funktsiooni tulemus peab olema kasutatav selles kontekstis, seega $S_2 <: T_2$
  - Aga argumentiga on vastupidi – argumendi tüüp peaks olema kasutatav meie funktsioonis, seega $T_1 <: S_1$

Alamtüüpimine

Syntax

t	::=		*terms:*
		x	*variable*
		λx:T.t	*abstraction*
		t t	*application*
v	::=		*values:*
		λx:T.t	*abstraction value*
T	::=		*types:*
		Top	*maximum type*
		T→T	*type of functions*
Γ	::=		*contexts:*
		∅	*empty context*
		Γ, x:T	*term variable binding*

Evaluation $\boxed{t \longrightarrow t'}$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-App2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

Alamtüüpimine ...

Subtyping $S <: T$

$$S <: S \quad \text{(S-Refl)}$$

$$\frac{S <: U \qquad U <: T}{S <: T} \quad \text{(S-Trans)}$$

$$S <: \mathsf{Top} \quad \text{(S-Top)}$$

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2} \quad \text{(S-Arrow)}$$

Typing $\Gamma \vdash t : T$

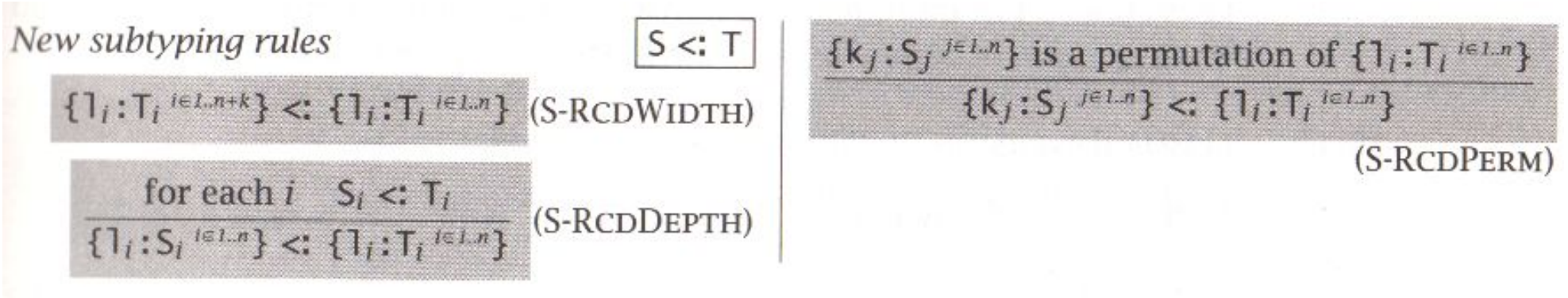
$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \text{(T-Var)}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1 . t_2 : T_1 \to T_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

$$\frac{\Gamma \vdash t : S \qquad S <: T}{\Gamma \vdash t : T} \quad \text{(T-Sub)}$$

Alamtüüpimine (kirjed)



Turvalisus

- Alamtüüpimine ilmselt ei mõjuta otseselt edenemise teoreemi
- Selleks, et tõestada säilitamise teoreemi, on meil vaja tõestada:
  - Alamtüüpimisrelatsiooni inversioonilemma
  - Inversioonilemma
  - Substitutsioonilemma

Top ja Bottom tüübid

- Top :> T
  - Ei pruugi olla alamtüübimisega keeles
  - Vastab Object-ile objekt-orienteeritud keeltes
- Bot <: T
  - Ei oma ühtegi väärtust
  - Mugav viis esitleda avaldisi, mis semantiliselt ei oma väärtusi, nt. erandid
  - Raskendab tüübikontrollija ehitamist

Tüübiomistus

- Up-cast
  - Omistame termile tema ülemtüübi
- Down-cast
  - Omistame termile suvalise teise tüübi, tüübikontrollija laseb läbi ja lisab täitmise-aja kontrolli
  - Kaotame edenemise
  - Kasutakse Javas selleks, et saada „poor-man's polymorphism“

Variantide alamtüüpimine

→ <> <:

Extends $\lambda_{<:}$ (15-1) and simple variant rules (11-11)

New syntactic forms

$t ::= \ldots$

$\langle l=t \rangle$ (no as) — *terms: tagging*

New typing rules

$\Gamma \vdash t : T$

$$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \langle l_1=t_1 \rangle : \langle l_1:T_1 \rangle}$$ (T-VARIANT)

New subtyping rules

$S <: T$

$$\langle l_i:T_i \; {}^{i\in 1..n} \rangle <: \langle l_i:T_i \; {}^{i\in 1..n+k} \rangle$$ (S-VARIANTWIDTH)

$$\frac{\text{for each } i \quad S_i <: T_i}{\langle l_i:S_i \; {}^{i\in 1..n} \rangle <: \langle l_i:T_i \; {}^{i\in 1..n} \rangle}$$ (S-VARIANTDEPTH)

$$\frac{\langle k_j:S_j \; {}^{j\in 1..n} \rangle \text{ is a permutation of } \langle l_i:T_i \; {}^{i\in 1..n} \rangle}{\langle k_j:S_j \; {}^{j\in 1..n} \rangle <: \langle l_i:T_i \; {}^{i\in 1..n} \rangle}$$ (S-VARIANTPERM)

Figure 15-5: Variants and subtyping

Viidete alamtüüpimine

- Viidete tüübid peavad olema ekvivalentsed: $\frac{S_1 <: T_1 \quad T_1 <: S_1}{Ref\ S_1 <: Ref\ T_1}$
- Tuleneb sellest, et me kasutame viiteid kirjutamiseks ja lugemiseks
- Sama käib massiivide kohta, kuid Javas on realiseeritud nii, et kui A <: B, siis A[] <: B[]
  - See toob sisse täitmise-aja kontrolli iga massiivi omistuse kohta

Alamtüüpimise probleemid

- Subsumtsioon
  - Reegli kasutamine ei ole määratud termi süntaksiga
  - Võime rakendada reegli mitu korda
  - Kaotame termi tüübi unikaalsuse
- Transitiivsus
  - Et reegli eelduses on ainult metamuutujad, saame rakendada reeglit mitu korda

Lahendus

- Algoritmiline alamtüüpimine
  - S <: T peab olema algoritmiliselt lahendatav
  - Viskame ära transitiivsuse ja refleksiivsuse
  - Ühendame kirjete reeglid üheks

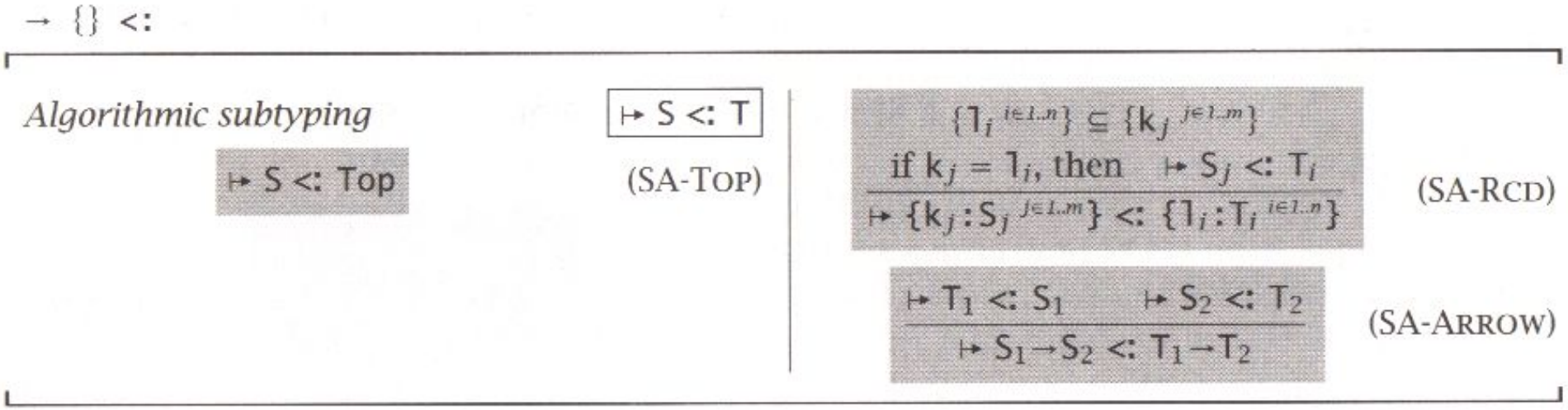


Figure 16-2: Algorithmic subtyping

Algoritmiline tüüpimine

- Peab olema algoritmiliselt lahendatav
- Peame muutma subsumtsiooni reeglit
  - Kus meil seda vaja on?
  - Selgub, et ainult aplikatsioonis
- Ülejäänud jäävad täpselt samaks

Lahendus ...

→ {} <:

Algorithmic typing $\boxed{\Gamma \mapsto t : T}$

$$\frac{x{:}T \in \Gamma}{\Gamma \mapsto x : T} \quad \text{(TA-VAR)}$$

$$\frac{\Gamma, x{:}T_1 \mapsto t_2 : T_2}{\Gamma \mapsto \lambda x{:}T_1 . t_2 : T_1 \to T_2} \quad \text{(TA-ABS)}$$

$$\frac{\Gamma \mapsto t_1 : T_1 \qquad T_1 = T_{11} \to T_{12} \qquad \Gamma \mapsto t_2 : T_2 \qquad \mapsto T_2 <: T_{11}}{\Gamma \mapsto t_1\ t_2 : T_{12}} \quad \text{(TA-APP)}$$

$$\frac{\text{for each } i \quad \Gamma \mapsto t_i : T_i}{\Gamma \mapsto \{l_1{=}t_1 \ldots l_n{=}t_n\} : \{l_1{:}T_1 \ldots l_n{:}T_n\}} \quad \text{(TA-RCD)}$$

$$\frac{\Gamma \mapsto t_1 : R_1 \qquad R_1 = \{l_1{:}T_1 \ldots l_n{:}T_n\}}{\Gamma \mapsto t_1 . l_i : T_i} \quad \text{(TA-PROJ)}$$

Figure 16-3: Algorithmic typing

Laiendused

- Hargnemine
  - On vaja tüübi ülemraja
  - Võib tulla vaja ka alarajast
- Bot <: T
  - Peame muuta applikatsioon nii, et kui Bot on funktsiooni tüüp, siis termi tüüp on ka Bot
  - Muutame ka kirjete projektsioon nii, et kui kirje tüüp on Bot tulemustüüp on ka Bot