Rewriting with strategies

- Global (innermost, outermost, etc.)
- Local (context-sensitive, etc.)
- Etc.

Benefits: constraints $\rightsquigarrow$ analysis finer

Drawbacks: constraints $\rightsquigarrow$ analysis more technical

Termination with strategies

innermost

Call by value

Innermost reduction

$$s \xrightarrow[R]{\Lambda}_{\mathsf{i}} t \text{ iff } \forall s_j \lhd s, s_j \text{ normalised}$$

- Pairs: $\langle u, v\rangle$ useless if redex in u
- Chains: restricted to innermost steps

$\{f(0,1,x) \to f(x,x,x), \quad \pi(x,y) \to x, \quad \pi(x,y) \to y\}$ innermost terminating

$$\langle \widehat{f}(0,1,x), \widehat{f}(x,x,x)\rangle \xrightarrow[R]{\neq\Lambda}_{\mathsf{i}}{}^{\star} \langle \widehat{f}(0,1,y), \widehat{f}(y,y,y)\rangle \text{ impossible}$$

Termination with strategies

innermost DP

Innermost chain: sequence $\langle s_i, t_i\rangle\ \langle s_{i+1}, t_{i+1}\rangle$ with

- σ normal ($\forall x \in \mathcal{X}, x\sigma$ in normal form)
- $t_i\sigma \xrightarrow[R]{\neq\Lambda}_{\mathsf{i}}{}^{\star} s_{i+1}\sigma$

Relation $s \twoheadrightarrow_{D,R} t$ if

- Proper sub-terms of s and t mortal
- σ normal such that $s \xrightarrow[R]{\neq\Lambda}_{\mathsf{i}}{}^{\star} s' \equiv u\sigma \xrightarrow[D]{\Lambda} t$
- Proper sub-terms of s' normalised

Theorem. (A. & G.)
$\to_R$ innermost terminating iff $\twoheadrightarrow_{\mathsf{DP}_i(R),R}$ innermost terminating

$$\mathsf{SN}_{\mathsf{inn}}(\to_R) \iff \mathsf{SN}_{\mathsf{inn}}(\twoheadrightarrow_{\mathsf{DP}_i(R),R})$$

Termination with strategies

innermost DP

Better? Yes!

Constraints on $\sigma \rightsquigarrow$ useless pairs, useless rules

- $\langle u, v\rangle$ and v without defined symbol $\rightsquigarrow$ 0 step
- $l \to r$ with redex proper subterm of $l \rightsquigarrow$ useless

Finer: for $G_i = \{\langle s_1, t_1\rangle, \ldots, \langle s_n, t_n\rangle\}$

$$\bigcup_{1\leq i,j\leq n} \{l \to r \in R | \exists \sigma, \tau, C \text{ such that } s_i\sigma, s_j\sigma \text{ norm., } t_i\sigma \to_{\mathsf{i}}{}^{\star} C[l\tau] \to_{\mathsf{i}} C[r\tau] \to_{\mathsf{i}}{}^{\star} s_j\sigma\}$$

Rk. still undecidable...

Termination with strategies — innermost UR

$\forall f \in \mathcal{F}, \mathsf{NR}(R,f) = \{l \to r \in R | l(\Lambda) = f \text{ and no redex } \lhd l\}$

Usables rules for t in R:

$$\begin{cases} \mathsf{UT}(R,x) = \emptyset \\ \mathsf{UT}(R, f(t_1,\dots,t_n)) = \mathsf{NR}(R,f) \cup \bigcup_{j=1}^{n} \mathsf{UT}(R \setminus \mathsf{NR}(R,f), t_j) \\ \qquad\qquad\qquad\qquad \cup \bigcup_{l\to r \in \mathsf{NR}(R,f)} \mathsf{UT}(R \setminus \mathsf{NR}(R,f), r) \end{cases}$$

Chains:

$$\langle s,t\rangle \xrightarrow[\mathsf{UT}(R,t)]{}_{\mathsf{i}}{}^{\star} \langle s',t'\rangle \xrightarrow[\mathsf{UT}(R,t')]{}_{\mathsf{i}}{}^{\star} \cdots$$

Termination with strategies — innermost DP

Theorem.
If $\forall G_i \subseteq G$ strongly connected $\exists(\geq, \succ)$ (wf, stable, w-mono where useful)

1. $\forall \langle s,t\rangle \in G_i, s \geq t$ and $\forall l \to r \in \mathsf{UT}(R,t), l \geq r$
2. $\exists \langle s_0,t_0\rangle \in G_i$ such that $s \succ t$

then $\mathsf{SN}_{\mathsf{inn}}(\twoheadrightarrow_{G,R})$

$$\begin{cases} x - 0 & \to & x \\ s(x) - s(y) & \to & x - y \\ 0 \div s(y) & \to & 0 \\ s(x) \div s(y) & \to & s((x-y) \div s(y)) \end{cases}$$

$\rightsquigarrow$ case of modular proof...

Termination with strategies — innermost graph

Graph approximation: $\langle s,t\rangle \overset{?}{\to}_{\mathsf{i}} \langle s',t'\rangle$

Proper sub-terms defined of t (possibly) rewritten but

- Not if sub-terms of s (already n.f.)
- Occurrences of variables not distinct (in σ hence n.f.)

$\rightsquigarrow \mathsf{CAP}(s,t)$

Unification not enough: $s\sigma, s'\sigma$ normal forms (for σ mgu)

Approx.: $\langle s,t\rangle$ and $\langle s',t'\rangle$ innermost connectable if

1. $\mathsf{mgu}(\mathsf{CAP}(s,t), s') = \mu$
2. $s\mu$ and $s'\mu$ normal forms

Termination with strategies — innermost graph

Theorem.
If arc in G from $\langle s,t\rangle$ to $\langle s',t'\rangle$ then innermost connectable

Proof induc. on t : $\exists\sigma | s\sigma$ n.f. and $t\sigma \to^{\star} u \Longrightarrow \exists\tau | \mathsf{CAP}(s,t)\sigma\tau = u$ (dom)

- $t \lhd s$ then $t\sigma$ n.f. $= u$, $\mathsf{CAP}(s,t) = t$ and $\tau = \emptyset$ OK
- $t \not\lhd s$ and $t(\Lambda) \in \mathcal{D}$ then $\mathsf{CAP}(s,t) = x$ fresh and $\tau = \{x \mapsto u\}$ OK
- $t \not\lhd s$ and $t = c(t_1,\dots,t_n)$ then
  $\mathsf{CAP}(s,t) = c(\mathsf{CAP}(s,t_1),\dots,\mathsf{CAP}(s,t_n))$
  $u = c(u_1,\dots,u_n)$ avec $t_j\sigma \to^{\star} u_j$
  By IH $\exists\tau_i | \mathsf{CAP}(s,t_j)\sigma\tau_j = u_j$, fresh var hence $\tau = \tau_1 \circ \cdots \circ \tau_n$ OK

Hence $\mathsf{CAP}(s,t)\sigma\tau = s'\sigma$ and by choice of domain $= s'\sigma\tau$ hence mgu + n.f.

Termination — innermost

Global benefit: classes where $\mathsf{SN}_{\mathsf{inn}}(\to_R) \Longrightarrow \mathsf{SN}(\to_R)$

(See Gramlich's works)

For automation, (coarse) test:
if no critical pair $\mathsf{SN}(\to_R) \Leftrightarrow \mathsf{SN}_{\mathsf{inn}}(\to_R)$)

(overlay locally confluent...)

$$\begin{cases} q(0, s(y), s(z)) & \to & 0 \\ q(s(x), s(y), z) & \to & q(x, y, z) \\ q(x, 0, s(z)) & \to & s(q(x, s(z), s(z))) \end{cases}$$

Rewriting — conditions

$l \to r \mid \underline{\{\cdots s_i \to t_i \cdots\}}_c$ list of conditions

- $V(r) \subseteq V(l) \cup V(c)$ — type 3
- $V(s_i) \subseteq V(l) \cup \bigcup_{j=1}^{i-1} V(t_j)$ — deterministic

Termination? $\rightsquigarrow$ operational termination

Rewriting — conditions

$l \to r \mid \underline{\{\cdots s_i \to t_i \cdots\}}_c$ list of conditions

- $V(r) \subseteq V(l) \cup V(c)$ — type 3
- $V(s_i) \subseteq V(l) \cup \bigcup_{j=1}^{i-1} V(t_j)$ — deterministic

Termination? $\rightsquigarrow$ operational termination

A transformation: n conditions into $c \rightsquigarrow n+1$ rules by U

$$\begin{aligned} \mathsf{U}(l \to r) &= \{l \to r\} \\ \mathsf{U}(l \to r \mid s \to t,\ c) &= \{l \to u(s, \vec{x})\} \cup \mathsf{U}(u(t, \vec{x}) \to r \mid c) \end{aligned}$$

with u new symbol, $\vec{x} = V(l) \cap (V(t) \cup V(r) \cup V(c))$

Sound, not complete (no computational equivalence)

```
fmod    LengthOfFiniteLists is
        sorts Nat NatList NatIList .      subsort NatList < NatIList .
        op 0  : -> Nat  .                 op s  : Nat -> Nat  .
        op zeros : -> NatIList .          op nil : -> NatList .
        op cons  : Nat NatIList -> NatIList   [strat (1 0)]  .
        op cons  : Nat NatList -> NatList   [strat (1 0)]  .
        op length  : NatList -> Nat  .
        vars M N : Nat .
        var IL : NatIList .
        var L : NatList .
        eq zeros = cons(0,zeros) .
        eq length(nil) = 0 .
        eq length(cons(N, L)) = s(length(L)) .
endfm
```



```
kind [Nat].    kind [NatIList] .
op 0 : -> [Nat] .    op nil : -> [NatList] .    op zeros : -> [NatIList] .
op s : [Nat] -> [Nat] .
op cons : [Nat] [NatIList] -> [NatIList]  [strat (1)]  .
op length : [NatIList] -> [Nat] .
mb 0  : Nat  .    mb zeros  : NatIList  .    mb nil  : NatList  .
cmb s(N)  : Nat  if N : Nat  .    cmb L  : NatIList  if L : NatList  .
cmb cons(N,IL)  : NatIList  if N : Nat ∧ IL : NatIList  .
cmb cons(N,L)  : NatList  if N : Nat ∧ L : NatList  .
cmb length(L)  : Nat  if L : NatList  .
eq zeros = cons(0,zeros) .
eq length(nil) = 0 .
ceq length(cons(N,L)) = s(length(L))  if N : Nat ∧ L : NatList  .
```



```
kind [Nat].    kind [NatIList] .
op 0 : -> [Nat] .    op nil : -> [NatList] .    op zeros : -> [NatIList] .
op s : [Nat] -> [Nat] .
op cons : [Nat] [NatIList] -> [NatIList] [strat (1)] .
op length : [NatIList] -> [Nat] .
mb 0  : Nat  .    mb zeros  : NatIList  .    mb nil  : NatList  .
cmb s(N)  : Nat  if N : Nat  .    cmb L  : NatIList  if L : NatList  .
cmb cons(N,IL)  : NatIList  if N : Nat ∧ IL : NatIList  .
cmb cons(N,L)  : NatList  if N : Nat ∧ L : NatList  .
cmb length(L)  : Nat  if L : NatList  .
eq zeros = cons(0,zeros) .
eq length(nil) = 0 .
ceq length(cons(N,L)) = s(length(L))  if N : Nat ∧ L : NatList  .
```



```
kind [Nat].    kind [NatIList] .
op 0 : -> [Nat] .    op nil : -> [NatList] .    op zeros : -> [NatIList] .
op s : [Nat] -> [Nat] .
op cons : [Nat] [NatIList] -> [NatIList]  [strat (1)]  .
op length : [NatIList] -> [Nat] .
mb 0 : Nat  .    mb zeros  : NatIList  .    mb nil  : NatList  .
cmb s(N)  : Nat  if N : Nat  .    cmb L  : NatIList  if L : NatList  .
cmb cons(N,IL)  : NatIList  if N : Nat ∧ IL : NatIList  .
cmb cons(N,L)  : NatList  if N : Nat ∧ L : NatList  .
cmb length(L)  : Nat  if L : NatList  .
eq zeros = cons(0,zeros) .
eq length(nil) = 0 .
ceq length(cons(N,L)) = s(length(L))  if N : Nat ∧ L : NatList  .
```

```
kind [Nat].    kind [NatIList] .
op 0 : -> [Nat] .    op nil : -> [NatList] .    op zeros : -> [NatIList] .
op s : [Nat] -> [Nat] .
op cons : [Nat] [NatIList] -> [NatIList]  [strat (1)]  .
op length : [NatIList] -> [Nat] .
mb 0  : Nat  .    mb zeros  : NatIList  .    mb nil  : NatList  .
cmb s(N)  : Nat  if N : Nat  .    cmb L  : NatIList  if L : NatList  .
cmb cons(N,IL)  : NatIList  if N : Nat ∧ IL : NatIList  .
cmb cons(N,L)  : NatList  if N : Nat ∧ L : NatList  .
cmb length(L)  : Nat  if L : NatList  .
eq zeros = cons(0,zeros) .
eq length(nil) = 0 .
ceq length(cons(N,L)) = s(length(L))  if N : Nat ∧ L : NatList  .
```



```
kind [Nat].    kind [NatIList] .
op 0 : -> [Nat] .    op nil : -> [NatList] .    op zeros : -> [NatIList] .
op s : [Nat] -> [Nat] .
op cons : [Nat] [NatIList] -> [NatIList]  [strat (1)]  .
op length : [NatIList] -> [Nat] .
mb 0  : Nat  .    mb zeros  : NatIList  .    mb nil  : NatList  .
cmb s(N)  : Nat  if N : Nat  .    cmb L  : NatIList  if L : NatList  .
cmb cons(N,IL)  : NatIList  if N : Nat ∧ IL : NatIList  .
cmb cons(N,L)  : NatList  if N : Nat ∧ L : NatList  .
cmb length(L)  : Nat  if L : NatList  .
eq zeros = cons(0,zeros) .
eq length(nil) = 0 .
ceq length(cons(N,L)) = s(length(L)) if N : Nat ∧ L : NatList .
```


Mainstream Programs: Membership & Conditions

T1 Sorts/memberships vers conditions

T2 Conditional into unconditional (unravelling)

} with CS

Termination of $(\text{T2} \circ \text{T1})(\mathcal{P}) \Rightarrow$ operational termination of $\mathcal{P}$


```
kind [Nat].    kind [NatIList] .
op 0 : -> [Nat] .    op nil : -> [NatList] .    op zeros : -> [NatIList] .
op s : [Nat] -> [Nat] .
op cons : [Nat] [NatIList] -> [NatIList]  [strat (1)]  .
op length : [NatIList] -> [Nat] .
mb 0  : Nat  .    mb zeros  : NatIList  .    mb nil  : NatList  .
cmb s(N)  : Nat  if N : Nat  .    cmb L  : NatIList  if L : NatList  .
cmb cons(N,IL)  : NatIList  if N : Nat ∧ IL : NatIList  .
cmb cons(N,L)  : NatList  if N : Nat ∧ L : NatList  .
cmb length(L)  : Nat  if L : NatList  .
eq zeros = cons(0,zeros) .
eq length(nil) = 0 .
ceq length(cons(N,L)) = s(length(L))  if N : Nat ∧ L : NatList  .
```

and(tt,T) -> T

isNat(0) -> tt

isNat(s(N)) -> isNat(N)

isNat(length(L)) -> isNatList(L)

isNatIList(IL) -> isNatList(IL)

isNatIList(zeros) -> tt

isNatIList(cons(N,IL)) -> and(isNat(N),isNatIList(IL))

isNatList(nil) -> tt

isNatList(cons(N,L)) -> and(isNat(N),isNatList(L))

zeros -> cons(0,zeros)

length(nil) -> 0

length(cons(N,L)) -> uLength(and(isNat(N),isNatList(L)),L)

uLength(tt,L) -> s(length(L))

$\mu(\text{isNat}) = \mu(\text{isNatList}) = \mu(\text{isNatIList}) = \emptyset$

$\mu(s) = \mu(\text{and}) = \mu(\text{cons}) = \mu(\text{length}) = \mu(\text{uLength}) = \{1\}$

Here pages of unreadable output from some automated prover...

```
 length(nil) -> 0
 uLength(tt,L) -> s(length(L))
 zeros -> cons(0,zeros)
-> Unhiding rules:
 Empty
->Strongly Connected Components:
 There is no strongly connected component

The problem is finite.
```

Trust me!

Rewriting techniques beyond pen & paper

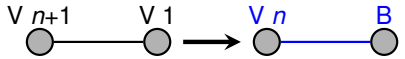
- Powerful techniques
- Automation

$\rightsquigarrow$ large/intricate tools $\rightsquigarrow$ bugs

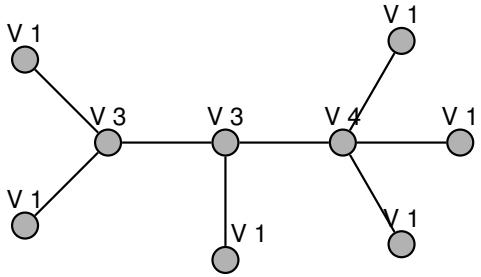
Question 1: Trust in automated provers?

Question 2: Automation in formal developments?

Rewriting — local interactions

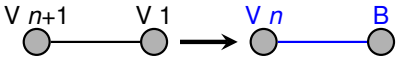


LC0 (+ cond.)

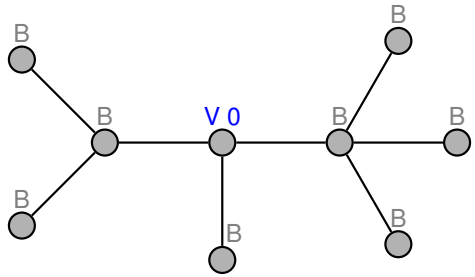


Conditions here: graph = tree $\wedge$ initial labels = node degrees

Rewriting — local interactions



LC0 (+ cond.)



Eventually. . . an elected node.