Symbolic Evaluation Graphs and Term Rewriting — A General Methodology for Analyzing Logic Programs

Jürgen Giesl

LuFG Informatik 2, RWTH Aachen University, Germany

joint work with T. Ströder, P. Schneider-Kamp, F. Emmes, and C. Fuhs

Termination Analysis for TRSs

$$\begin{array}{lrcl} \mathcal{R}: & \mathsf{double}(0) & \rightarrow & 0 \\ & \mathsf{double}(\mathsf{s}(x)) & \rightarrow & \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \end{array}$$

$\mathcal{R}$ is *terminating* iff there is no infinite evaluation $t_1 \rightarrow_{\mathcal{R}} t_2 \rightarrow_{\mathcal{R}} \ldots$

Computation of "double(1)":

$$\begin{array}{rcl} \mathsf{double}(\mathsf{s}(0)) & \rightarrow_{\mathcal{R}} & \mathsf{s}(\mathsf{s}(\mathsf{double}(0))) \\ & \rightarrow_{\mathcal{R}} & \mathsf{s}(\mathsf{s}(0)) \end{array}$$

- easier / more general than for programs
- suitable for automation
- **But:** halting problem is undecidable!
  $\Rightarrow$ automated termination proofs do not always succeed

Automated Termination Tools for TRSs

- AProVE *(Aachen)*
- CARIBOO *(Nancy)*
- CiME *(Orsay)*
- Jambox *(Amsterdam)*
- Matchbox *(Leipzig)*
- MU-TERM *(Valencia)*
- MultumNonMulta *(Kassel)*
- TEPARLA *(Eindhoven)*
- Termptation *(Barcelona)*
- TORPA *(Eindhoven)*
- TPA *(Eindhoven)*
- TTT *(Innsbruck)*
- VMTL *(Vienna)*

- Annual *International Competition of Termination Tools*
- well-developed field
- active research
- powerful techniques & tools
- **But:** What about application in practice?

Termination of Programming Languages

Functional Languages

- first-order languages with strict evaluation strategy
  (Walther, 94), (Giesl, 95), (Lee, Jones, Ben-Amram, 01)
- ensuring termination (e.g., by typing)
  (Telford & Turner, 00), (Xi, 02), (Abel, 04), (Barthe et al, 04) etc.
- outermost termination of untyped first-order rewriting
  (Fissore, Gnaedig, Kirchner, 02), (Endrullis & Hendriks, 09), (Raffelsieper & Zantema, 09), (Thiemann, 09)
- automated technique for small HASKELL-like language
  (Panitz & Schmidt-Schauss, 97)

- do not work on full existing languages
- no use of TRS-techniques (stand-alone methods)

Termination of Programming Languages

Functional Languages

- using TRS-techniques for HASKELL is challenging
  - HASKELL has a lazy evaluation strategy.
    For TRSs, one proves termination of *all* reductions.
  - HASKELL's equations are handled from top to bottom.
    For TRSs, *any* rule may be used for rewriting.
  - HASKELL has polymorphic types.
    TRSs are *untyped*.
  - In HASKELL-programs, often only some functions terminate.
    TRS-methods try to prove termination of *all* terms.
  - HASKELL is a higher-order language.
    Most automatic TRS-methods only handle *first-order* rewriting.

Termination of Programming Languages

Functional Languages

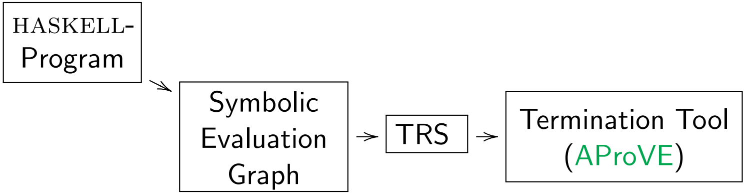
- using TRS-techniques for HASKELL is challenging
- **New approach** (ACM TOPLAS '11)
  - Frontend
    - evaluate HASKELL a few steps $\Rightarrow$ symbolic evaluation graph

      graph captures evaluation strategy, types, etc.
    - transform symbolic evaluation graph $\Rightarrow$ TRS
  - Backend
    - prove termination of the resulting TRS
      (using existing techniques & tools)
- implemented in **AProVE**
  - accepts full HASKELL 98 language
  - successfully evaluated with standard HASKELL-libraries
    (succeeds on approx. 80 % of the functions in standard libraries)

Termination of Programming Languages

Functional Languages

- using TRS-techniques for HASKELL is challenging



- implemented in **AProVE**
  - accepts full HASKELL 98 language
  - successfully evaluated with standard HASKELL-libraries
    (succeeds on approx. 80 % of the functions in standard libraries)

Termination of Programming Languages

Imperative Languages

- Synthesis of Linear Ranking Functions
  (Colon & Sipma, 01), (Podelski & Rybalchenko, 04)
- Terminator: Termination Analysis by Abstraction & Model Checking
  (Cook, Podelski, Rybalchenko et al., since 05)
- Julia & COSTA: Termination Analysis of JAVA BYTECODE
  (Spoto, Mesnard, Payet, 10),
  (Albert, Arenas, Codish, Genaim, Puebla, Zanardini, 08)
- ...

- used at Microsoft for verifying Windows device drivers
- no use of TRS-techniques (stand-alone methods)

Termination of Programming Languages

Imperative Languages

- using TRS-techniques for JAVA is challenging
  - sharing and aliasing
  - side effects
  - cyclic data objects
  - object-orientation
  - recursion
  - ...

Termination of Programming Languages

Imperative Languages

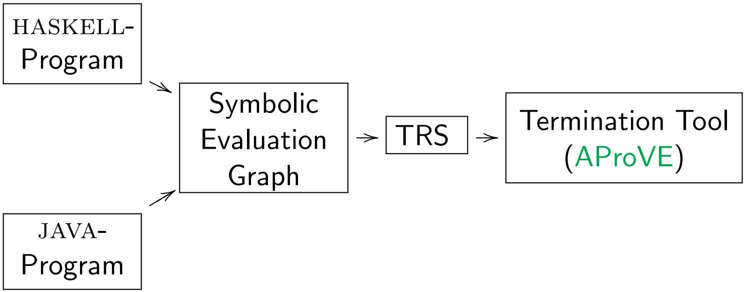
- using TRS-techniques for JAVA is challenging
- **New approach** (RTA '10, RTA '11, CAV '12)
  - Frontend
    - evaluate JAVA a few steps $\Rightarrow$ symbolic evaluation graph

      graph captures side effects, sharing, cyclic data objects, etc.
    - transform symbolic evaluation graph $\Rightarrow$ TRS
  - Backend
    - prove termination of the resulting TRS
      (using existing techniques & tools)
- implemented in **AProVE**
  - successfully evaluated on JAVA-collection
  - most powerful termination tool for JAVA
    (winner of the international *termination competition* for JAVA)

Termination of Programming Languages

Imperative Languages

- using TRS-techniques for JAVA is challenging



- implemented in **AProVE**
  - successfully evaluated on JAVA-collection
  - most powerful termination tool for JAVA (winner of the international *termination competition* for JAVA)

Termination of Programming Languages

Logic Languages

- well-developed field *(De Schreye & Decorte, 94) etc.*
- **direct approaches:** work directly on the logic program
  - cTI *(Mesnard et al)*
  - TerminWeb *(Codish et al)*
  - TermiLog *(Lindenstrauss et al)*
  - Polytool *(Nguyen, De Schreye, Giesl, Schneider-Kamp)*

  TRS-techniques can be adapted to work *directly* on the LP
- **transformational approaches:** transform LP to TRS

$\mathsf{app}([\,], YS, YS).$
$\mathsf{app}([X \mid XS], YS, [X \mid ZS]) \text{ :- } \mathsf{app}(XS, YS, ZS).$

$$\begin{aligned} \mathsf{app}^{in}([\,], YS) &\to \mathsf{app}^{out}(YS) \\ \mathsf{app}^{in}([X \mid XS], YS) &\to \mathsf{u}(\mathsf{app}^{in}(XS, YS), X) \\ \mathsf{u}(\mathsf{app}^{out}(ZS), X) &\to \mathsf{app}^{out}([X \mid ZS]) \end{aligned}$$

- **class of queries** $\mathcal{Q}_m^{\mathsf{p}}$ described by *predicate* p and *moding* m
  Example: $\mathcal{Q}_m^{\mathsf{app}} = \{\mathsf{app}(t_1, t_2, t_3) \mid t_1, t_2 \text{ are ground}\}$.

- encode **atom** $p(\ldots)$ to **terms** $p^{in}(\ldots)$, $p^{out}(\ldots)$
  - arguments of p^{in}: input arguments of $p(\ldots)$
  - arguments of p^{out}: remaining arguments of $p(\ldots)$

Encoding of $\mathsf{app}([\,], YS, YS)$:	$\mathsf{app}^{in}([\,], YS)$,	$\mathsf{app}^{out}(YS)$
Encoding of $\mathsf{app}([X \mid XS], YS, [X \mid ZS])$:	$\mathsf{app}^{in}([X \mid XS], YS)$,	$\mathsf{app}^{out}([X \mid ZS])$
Encoding of $\mathsf{app}(XS, YS, ZS)$:	$\mathsf{app}^{in}(XS, YS)$,	$\mathsf{app}^{out}(ZS)$

- encode **clauses** to **rewrite rules**
  - fact $p(\ldots)$: $\quad p^{in}(\ldots) \to p^{out}(\ldots)$
  - rule $p(\ldots) \text{ :- } q(\ldots)$: $\quad p^{in}(\ldots) \to \mathsf{u}(q^{in}(\ldots))$

$$\mathsf{u}(q^{out}(\ldots)) \to p^{out}(\ldots)$$

Termination of Programming Languages

Logic Languages

- well-developed field *(De Schreye & Decorte, 94) etc.*
- **direct approaches:** work directly on the logic program
  - cTI *(Mesnard et al)*
  - TerminWeb *(Codish et al)*
  - TermiLog *(Lindenstrauss et al)*
  - Polytool *(Nguyen, De Schreye, Giesl, Schneider-Kamp)*

  TRS-techniques can be adapted to work *directly* on the LP
- **transformational approaches:** transform LP to TRS
  - TALP *(Ohlebusch et al)*
  - AProVE *(Giesl et al)*
- only for *definite* LP (without cut)
- not for real PROLOG

Termination of Programming Languages

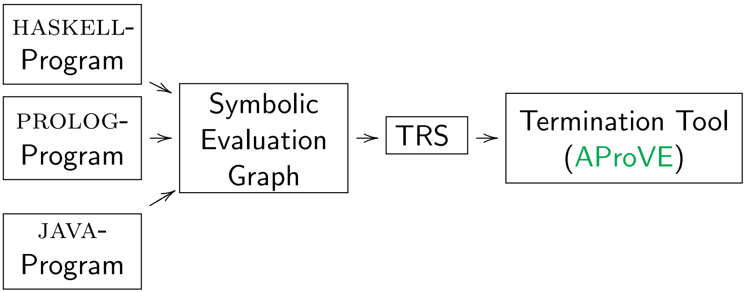
Logic Languages

- analyzing PROLOG is challenging due to cuts etc.
- **New approach**
  - Frontend
    - evaluate PROLOG a few steps $\Rightarrow$ symbolic evaluation graph

      graph captures evaluation strategy due to cuts etc.
    - transform symbolic evaluation graph $\Rightarrow$ TRS
  - Backend
    - prove termination of the resulting TRS
      (using existing techniques & tools)
- implemented in **AProVE**
  - successfully evaluated on PROLOG-collections with cuts
  - most powerful termination tool for PROLOG
    (winner of *termination competition* for PROLOG)

Logic Languages

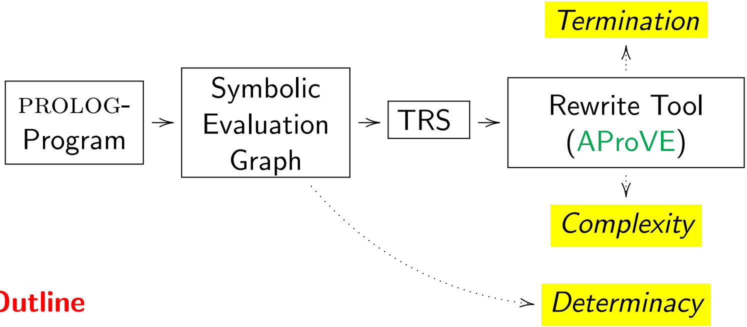
- analyzing PROLOG is challenging due to cuts etc.



- implemented in **AProVE**
  - successfully evaluated on PROLOG-collections with cuts
  - most powerful termination tool for PROLOG
    (winner of *termination competition* for PROLOG)

Symbolic Evaluation Graphs and Term Rewriting

General methodology for analyzing PROLOG programs



Outline

- linear operational semantics of PROLOG
- from PROLOG to symbolic evaluation graphs
- from symbolic evaluation graphs to TRSs for termination analysis
- from symbolic evaluation graphs to TRSs for complexity analysis
- determinacy analysis

$$
\begin{array}{rll}
\mathsf{star}(XS, [\,]) & \text{:- } !. & (1)\\
\mathsf{star}([\,], ZS) & \text{:- } !, \mathsf{eq}(ZS, [\,]). & (2)\\
\mathsf{star}(XS, ZS) & \text{:- } \mathsf{app}(XS, YS, ZS), \mathsf{star}(XS, YS). & (3)\\
\mathsf{app}([\,], YS, YS). & & (4)\\
\mathsf{app}([X \mid XS], YS, [X \mid ZS]) & \text{:- } \mathsf{app}(XS, YS, ZS). & (5)\\
\mathsf{eq}(X, X). & & (6)
\end{array}
$$

- $\mathsf{star}(t_1, t_2)$ holds iff t_2 results from concatenation of t_1 $\quad (t_2 \in (t_1)^*)$
  - $\mathsf{star}([1, 2], [\,])$ holds
  - $\mathsf{star}([1, 2], [1, 2])$ holds, since $\mathsf{app}([1, 2], [\,], [1, 2]), \mathsf{star}([1, 2], [\,])$ hold
  - $\mathsf{star}([1, 2], [1, 2, 1, 2])$ holds, etc.

- cut in clause (2) needed for *termination*. Otherwise:

  $\mathsf{star}([\,], t)$ would lead to

  $\mathsf{app}([\,], YS, t), \mathsf{star}([\,], YS)$ would lead to

  $\mathsf{star}([\,], t)$

$$
\begin{aligned}
\mathsf{star}(XS, [\,]) &\ \texttt{:-}\ !. && (1)\\
\mathsf{star}([\,], ZS) &\ \texttt{:-}\ !, \mathsf{eq}(ZS, [\,]). && (2)\\
\mathsf{star}(XS, ZS) &\ \texttt{:-}\ \mathsf{app}(XS, YS, ZS), \mathsf{star}(XS, YS). && (3)\\
\mathsf{app}([\,], YS, YS). & && (4)\\
\mathsf{app}([X \mid XS], YS, [X \mid ZS]) &\ \texttt{:-}\ \mathsf{app}(XS, YS, ZS). && (5)\\
\mathsf{eq}(X, X). & && (6)
\end{aligned}
$$

- **state:** $(G_1 \mid \ldots \mid G_n)$ with current goal G_1 and next goals $G_2, \ldots, G_n$
- **goal:** $(t_1, \ldots, t_k)$ query or

  $(t_1, \ldots, t_k)^c$ query labeled by clause c used for next resolution
- **inference rules:**
  - Case
  - Eval
  - Back
  - Cut
  - Suc

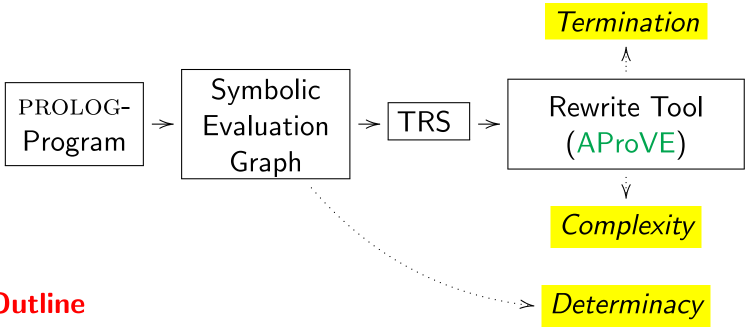
$$
\begin{array}{rl}
\mathsf{star}([1,2],[\,]) & \vdash_{\textsc{Case}}\\
\mathsf{star}([1,2],[\,])^{(1)} \mid \mathsf{star}([1,2],[\,])^{(2)} \mid \mathsf{star}([1,2],[\,])^{(3)} & \vdash_{\textsc{Eval}}\\
! \mid \mathsf{star}([1,2],[\,])^{(2)} \mid \mathsf{star}([1,2],[\,])^{(3)} & \vdash_{\textsc{Cut}}\\
\square & \vdash_{\textsc{Suc}}\\
\varepsilon &
\end{array}
$$

$$
\begin{array}{rll}
\mathsf{star}(XS, [\,]) & \text{:-}\ !. & (1)\\
\mathsf{star}([\,], ZS) & \text{:-}\ !, \mathsf{eq}(ZS, [\,]). & (2)\\
\mathsf{star}(XS, ZS) & \text{:-}\ \mathsf{app}(XS, YS, ZS), \mathsf{star}(XS, YS). & (3)\\
\mathsf{app}([\,], YS, YS). & & (4)\\
\mathsf{app}([X \mid XS], YS, [X \mid ZS]) & \text{:-}\ \mathsf{app}(XS, YS, ZS). & (5)\\
\mathsf{eq}(X, X). & & (6)
\end{array}
$$

- **state:** $(G_1 \mid \ldots \mid G_n)$ with current goal G_1 and next goals $G_2, \ldots, G_n$
- *linear* semantics, since state contains all backtracking information
  $\Rightarrow$ evaluation is a sequence of states, not a search tree
- suitable for extension to abstract states

$$
\begin{array}{rl}
\mathsf{star}([1,2],[\,]) & \vdash_{\textsc{Case}}\\
\mathsf{star}([1,2],[\,])^{(1)} \mid \mathsf{star}([1,2],[\,])^{(2)} \mid \mathsf{star}([1,2],[\,])^{(3)} & \vdash_{\textsc{Eval}}\\
! \mid \mathsf{star}([1,2],[\,])^{(2)} \mid \mathsf{star}([1,2],[\,])^{(3)} & \vdash_{\textsc{Cut}}\\
\square & \vdash_{\textsc{Suc}}\\
\varepsilon &
\end{array}
$$

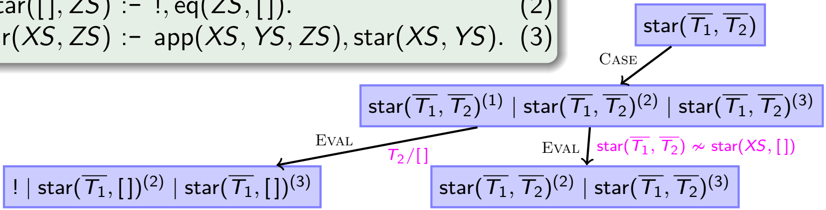
General methodology for analyzing PROLOG programs



Outline

- linear operational semantics of PROLOG
- from PROLOG to symbolic evaluation graphs
- from symbolic evaluation graphs to TRSs for termination analysis
- from symbolic evaluation graphs to TRSs for complexity analysis
- determinacy analysis

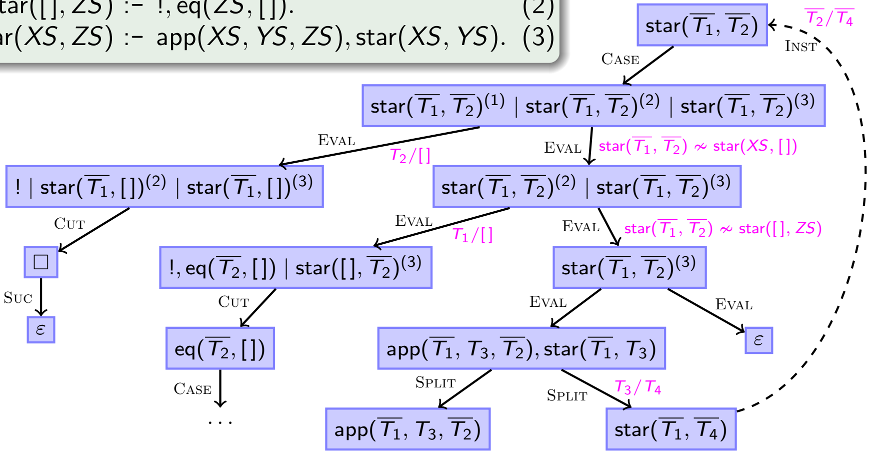
$$
\begin{array}{lr}
\color{red}{\mathsf{star}(XS, [\,]) \;\text{:-}\; !.} & \color{red}{(1)}\\
\mathsf{star}([\,], ZS) \;\text{:-}\; !, \mathsf{eq}(ZS, [\,]). & (2)\\
\mathsf{star}(XS, ZS) \;\text{:-}\; \mathsf{app}(XS, YS, ZS), \mathsf{star}(XS, YS). & (3)
\end{array}
$$



- **symbolic evaluation graph:** all evaluations for a *class* of queries
- **class of queries** $\mathcal{Q}_m^{\mathsf{p}}$ described by *predicate p* and *moding m*

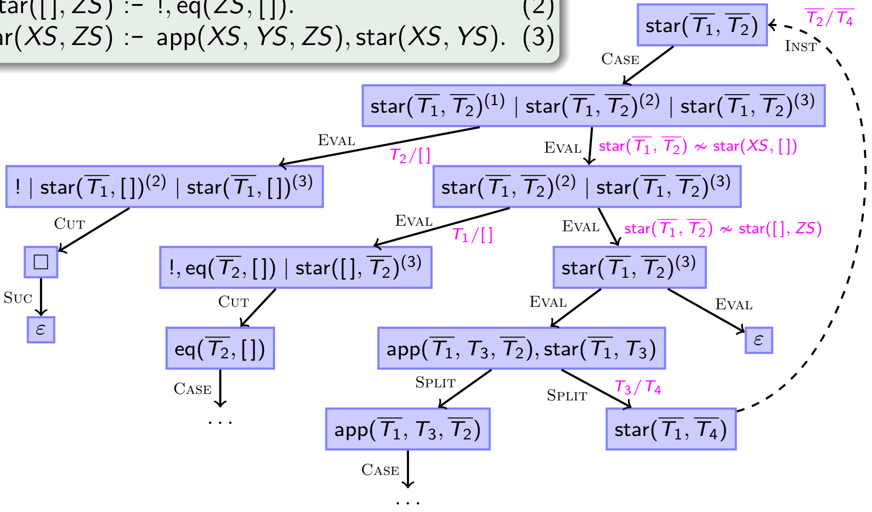
  Example: $\mathcal{Q}_m^{\mathsf{star}} = \{\mathsf{star}(t_1, t_2) \mid t_1, t_2 \text{ are ground}\}$.
- **abstract state:** stands for *set* of concrete states
  - state with *abstract* variables $T_1, T_2, \ldots$ representing arbitrary terms
  - constraints on the terms represented by $T_1, T_2, \ldots$
    - groundness constraints: $\overline{T_1}$, $\overline{T_2}$
    - unification constraints: $\mathsf{star}(\overline{T_1}, \overline{T_2}) \nsim \mathsf{star}(XS, [\,])$

$$\mathsf{star}(XS, [\,]) \;\text{:-}\; !. \qquad (1)$$
$$\mathsf{star}([\,], ZS) \;\text{:-}\; !, \mathsf{eq}(ZS, [\,]). \qquad (2)$$
$$\mathsf{star}(XS, ZS) \;\text{:-}\; \mathsf{app}(XS, YS, ZS), \mathsf{star}(XS, YS). \qquad (3)$$



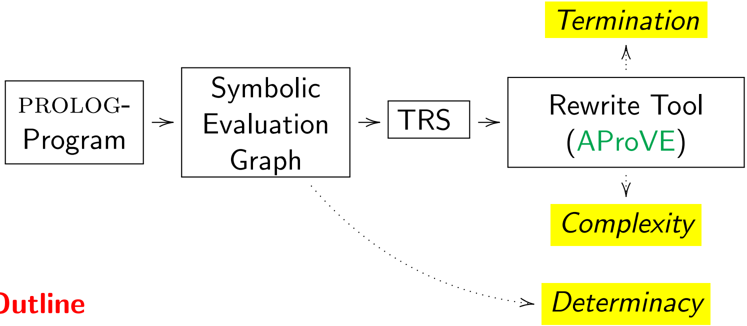
- Inst: connection to previous state if current state is an *instance*
- Split: split away first atom from a query
  - fresh variables in Split's second successor
  - approximate first atom's answer substitution by *groundness analysis*

$$
\begin{array}{rll}
\mathsf{star}(XS, [\,]) & \text{:- } !. & (1)\\
\mathsf{star}([\,], ZS) & \text{:- } !, \mathsf{eq}(ZS, [\,]). & (2)\\
\mathsf{star}(XS, ZS) & \text{:- } \mathsf{app}(XS, YS, ZS), \mathsf{star}(XS, YS). & (3)
\end{array}
$$



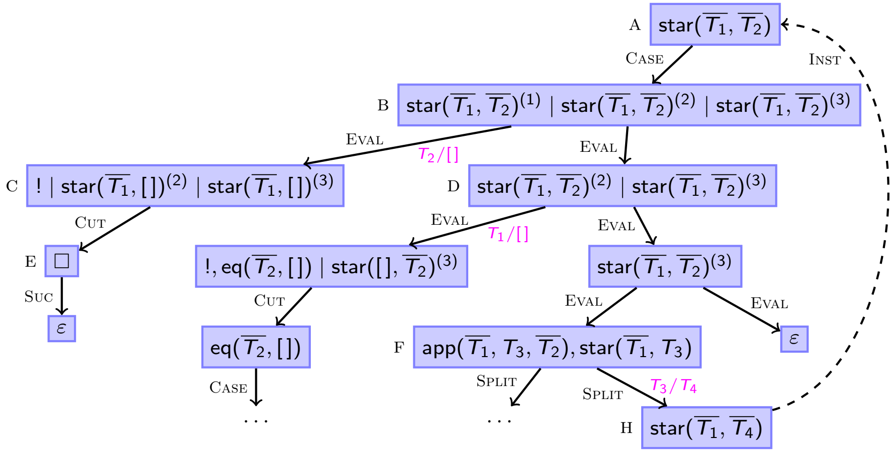
Symbolic Evaluation Graphs and Term Rewriting

General methodology for analyzing PROLOG programs



Outline

- linear operational semantics of PROLOG
- from PROLOG to symbolic evaluation graphs
- from symbolic evaluation graphs to TRSs for termination analysis
- from symbolic evaluation graphs to TRSs for complexity analysis
- determinacy analysis



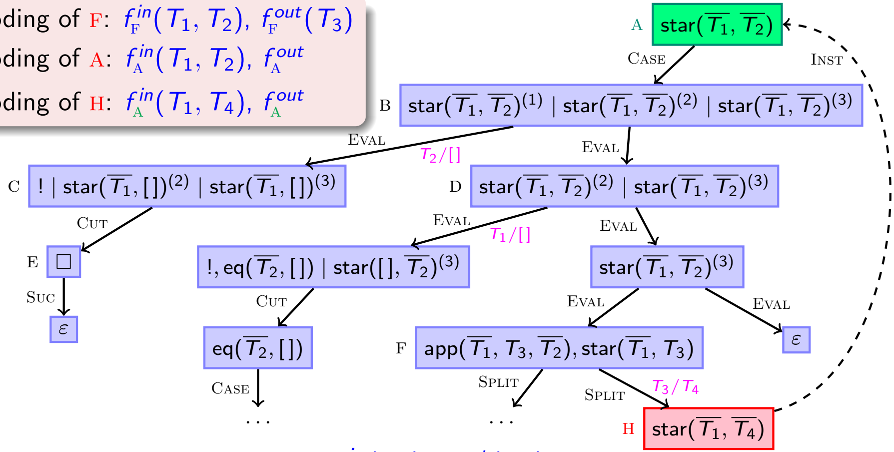
- **Aim:** show termination of concrete states represented by graph
- **Solution:** synthesize TRS from the graph
  - TRS captures all evaluations that are crucial for termination behavior
  - existing rewrite tools can show termination of TRS

    $\Rightarrow$ prove termination of original PROLOG program

Encoding of F: $f_{\text{F}}^{in}(T_1, T_2),\ f_{\text{F}}^{out}(T_3)$

Encoding of A: $f_{\text{A}}^{in}(T_1, T_2),\ f_{\text{A}}^{out}$

Encoding of H: $f_{\text{A}}^{in}(T_1, T_4),\ f_{\text{A}}^{out}$

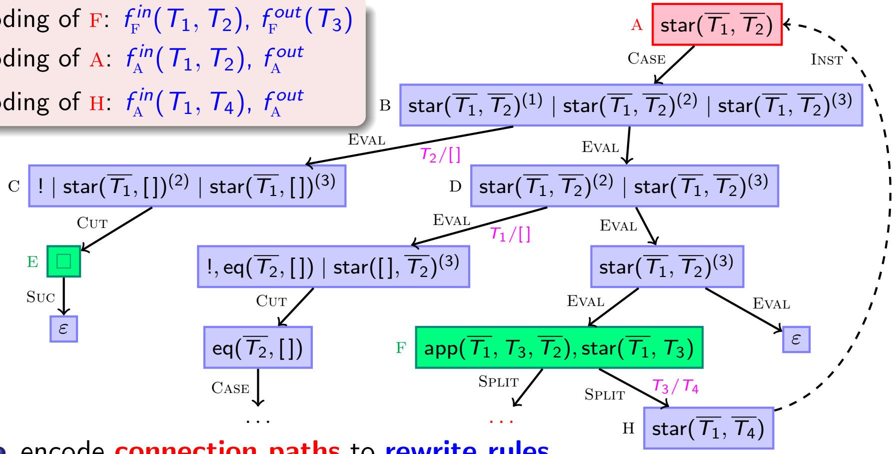


- encode **state** s to **terms** $f_s^{in}(\ldots),\ f_s^{out}(\ldots)$
  - arguments of f_s^{in}: abstract ground variables of s $(\overline{T_1}, \overline{T_2}, \ldots)$
  - arguments of f_s^{out}: remaining abstract variables of s which are made ground by every answer substitution of s (*groundness analysis*)
- for state s with INST edge to s': use $f_{s'}^{in},\ f_{s'}^{out}$ instead of $f_s^{in},\ f_s^{out}$

Encoding of F: $f_{\mathrm{F}}^{in}(T_1, T_2),\ f_{\mathrm{F}}^{out}(T_3)$

Encoding of A: $f_{\mathrm{A}}^{in}(T_1, T_2),\ f_{\mathrm{A}}^{out}$

Encoding of H: $f_{\mathrm{A}}^{in}(T_1, T_4),\ f_{\mathrm{A}}^{out}$

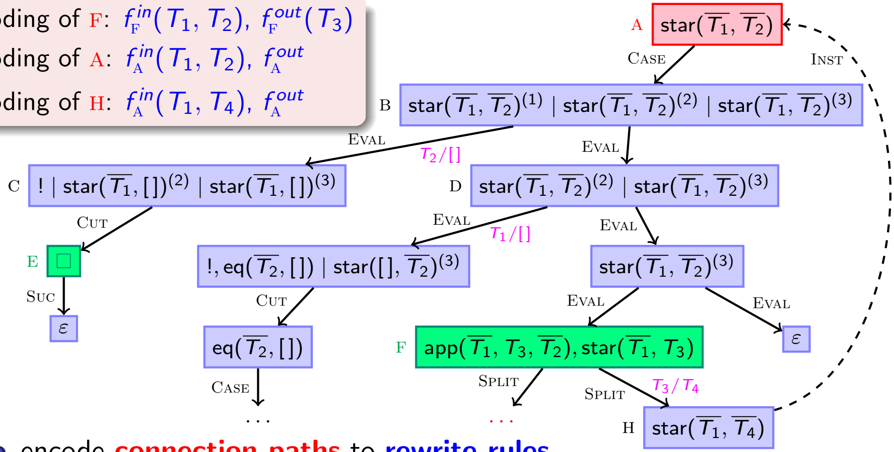


- encode **connection paths** to **rewrite rules**
- **connection path**:
  - start state = root, successor of Inst, or successor of Split but no Inst or Split node itself
  - end state = Inst, Split, Suc node, or successor of Inst node
  - connection path may not traverse end nodes except Suc nodes

Encoding of F: $f_{\mathrm{F}}^{in}(T_1, T_2),\ f_{\mathrm{F}}^{out}(T_3)$

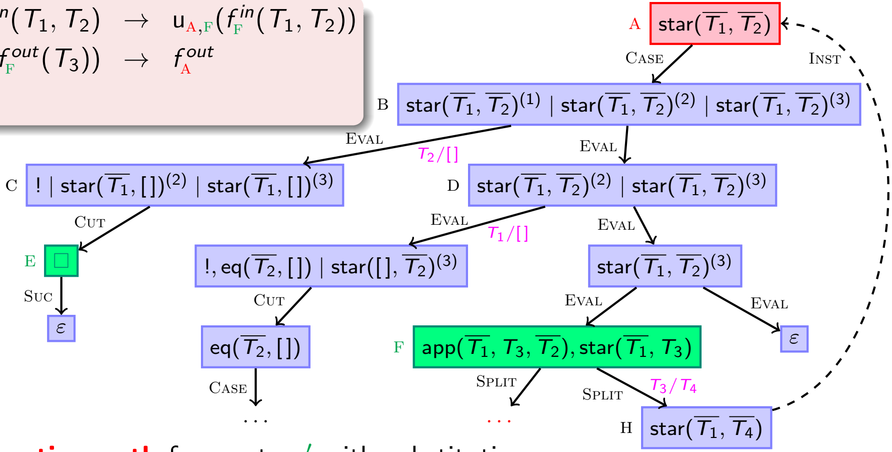
Encoding of A: $f_{\mathrm{A}}^{in}(T_1, T_2),\ f_{\mathrm{A}}^{out}$

Encoding of H: $f_{\mathrm{A}}^{in}(T_1, T_4),\ f_{\mathrm{A}}^{out}$



- encode **connection paths** to **rewrite rules**
- **connection path**: cover all ways through graph except
  - INST edges (are covered by the encoding of terms)
  - SPLIT edges (will be covered by extra SPLIT rules later)
  - parts without cycles or SUC nodes (irrelevant for termination behavior)

$$
\begin{aligned}
f^{in}_{\mathrm{A}}(T_1, T_2) &\rightarrow \mathsf{u}_{\mathrm{A},\mathrm{F}}(f^{in}_{\mathrm{F}}(T_1, T_2)) \\
\mathsf{u}_{\mathrm{A},\mathrm{F}}(f^{out}_{\mathrm{F}}(T_3)) &\rightarrow f^{out}_{\mathrm{A}}
\end{aligned}
$$



connection path from s to s' with substitution σ:

$f^{in}_{s}(\ldots)\sigma$ evaluates to $f^{out}_{s}(\ldots)\sigma$ if
$f^{in}_{s'}(\ldots)$ evaluates to $f^{out}_{s'}(\ldots)$

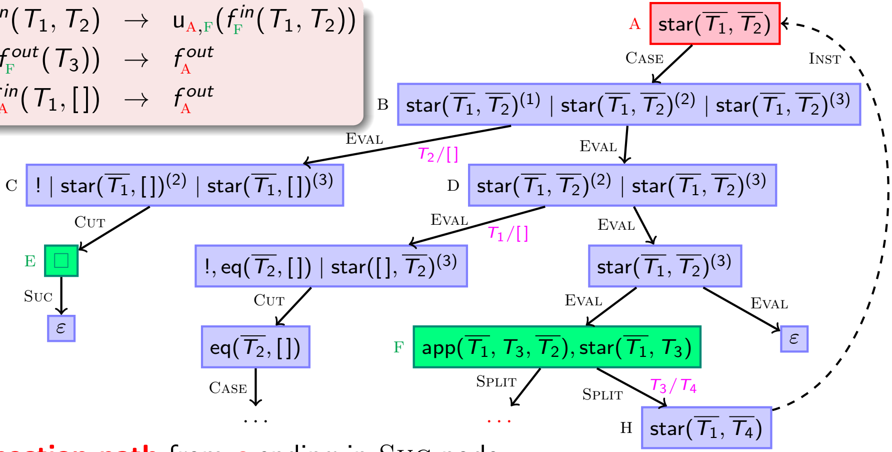
$f^{in}_{\mathrm{A}}(T_1, T_2)$ evaluates to f^{out}_{A} if
$f^{in}_{\mathrm{F}}(T_1, T_2)$ evaluates to $f^{out}_{\mathrm{F}}(T_3)$

rewrite rules:

$$
\begin{aligned}
f^{in}_{s}(\ldots)\sigma &\rightarrow \mathsf{u}_{s,s'}(\ f^{in}_{s'}(\ldots)\ ) \\
\mathsf{u}_{s,s'}(\ f^{out}_{s'}(\ldots)\ ) &\rightarrow f^{out}_{s}(\ldots)\sigma
\end{aligned}
$$

$$
\begin{aligned}
f^{in}_{\mathrm{A}}(T_1, T_2) &\rightarrow \mathsf{u}_{\mathrm{A},\mathrm{F}}(\ f^{in}_{\mathrm{F}}(T_1, T_2)\ ) \\
\mathsf{u}_{\mathrm{A},\mathrm{F}}(\ f^{out}_{\mathrm{F}}(T_3)\ ) &\rightarrow f^{out}_{\mathrm{A}}
\end{aligned}
$$

$$
\begin{aligned}
f^{in}_{\mathrm{A}}(T_1, T_2) &\rightarrow \mathsf{u}_{\mathrm{A},\mathrm{F}}(f^{in}_{\mathrm{F}}(T_1, T_2)) \\
\mathsf{u}_{\mathrm{A},\mathrm{F}}(f^{out}_{\mathrm{F}}(T_3)) &\rightarrow f^{out}_{\mathrm{A}} \\
f^{in}_{\mathrm{A}}(T_1, []) &\rightarrow f^{out}_{\mathrm{A}}
\end{aligned}
$$



connection path from s ending in SUC node:

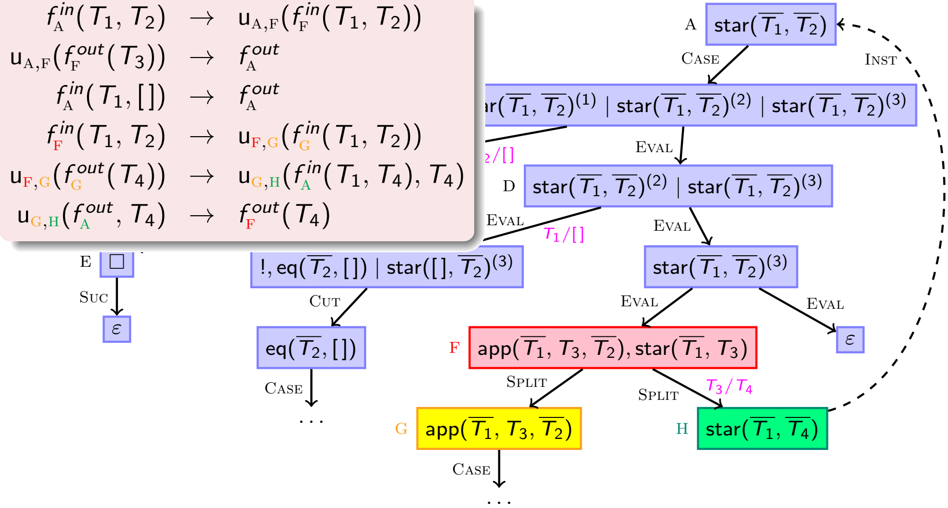
$f^{in}_{s}(\ldots)\sigma$ evaluates to $f^{out}_{s}(\ldots)\sigma$ $\qquad$ $f^{in}_{\mathrm{A}}(T_1, [])$ evaluates to f^{out}_{A}

intuition:

$f^{in}_{\mathrm{A}}(T_1, T_2)$ evaluates to f^{out}_{A} $\qquad$ if $T_2 \in (T_1)^*$

$f^{in}_{\mathrm{F}}(T_1, T_2)$ evaluates to $f^{out}_{\mathrm{F}}(T_3)$ $\qquad$ if $T_1 \neq []$, $T_2 \neq []$, T_3 is T_2 without prefix T_1, $T_3 \in (T_1)^*$

$$
\begin{aligned}
f^{in}_{\mathrm{A}}(T_1, T_2) &\rightarrow \mathsf{u}_{\mathrm{A,F}}(f^{in}_{\mathrm{F}}(T_1, T_2))\\
\mathsf{u}_{\mathrm{A,F}}(f^{out}_{\mathrm{F}}(T_3)) &\rightarrow f^{out}_{\mathrm{A}}\\
f^{in}_{\mathrm{A}}(T_1, []) &\rightarrow f^{out}_{\mathrm{A}}\\
f^{in}_{\mathrm{F}}(T_1, T_2) &\rightarrow \mathsf{u}_{\mathrm{F,G}}(f^{in}_{\mathrm{G}}(T_1, T_2))\\
\mathsf{u}_{\mathrm{F,G}}(f^{out}_{\mathrm{G}}(T_4)) &\rightarrow \mathsf{u}_{\mathrm{G,H}}(f^{in}_{\mathrm{A}}(T_1, T_4), T_4)\\
\mathsf{u}_{\mathrm{G,H}}(f^{out}_{\mathrm{A}}, T_4) &\rightarrow f^{out}_{\mathrm{F}}(T_4)
\end{aligned}
$$

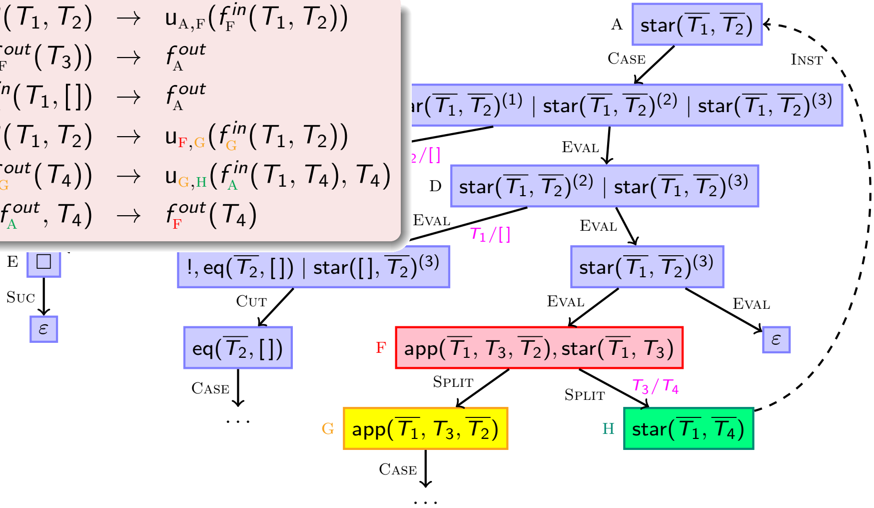


SPLIT node s with successors s_1 and s_2:

$f^{in}_{s}(\ldots)\sigma$ evaluates to $f^{out}_{s}(\ldots)\sigma$ if
$f^{in}_{s_1}(\ldots)\sigma$ evaluates to $f^{out}_{s_1}(\ldots)\sigma$ and
$f^{in}_{s_2}(\ldots)$ evaluates to $f^{out}_{s_2}(\ldots)$

$f^{in}_{\mathrm{F}}(T_1, T_2)$ evaluates to $f^{out}_{\mathrm{F}}(T_4)$ if
$f^{in}_{\mathrm{G}}(T_1, T_2)$ evaluates to $f^{out}_{\mathrm{G}}(T_4)$ and
$f^{in}_{\mathrm{A}}(T_1, T_4)$ evaluates to f^{out}_{A}

$$
\begin{aligned}
f^{in}_{\mathrm{A}}(T_1, T_2) &\to u_{\mathrm{A},\mathrm{F}}(f^{in}_{\mathrm{F}}(T_1, T_2))\\
u_{\mathrm{A},\mathrm{F}}(f^{out}_{\mathrm{F}}(T_3)) &\to f^{out}_{\mathrm{A}}\\
f^{in}_{\mathrm{A}}(T_1, [\,]) &\to f^{out}_{\mathrm{A}}\\
f^{in}_{\mathrm{F}}(T_1, T_2) &\to u_{\mathrm{F},\mathrm{G}}(f^{in}_{\mathrm{G}}(T_1, T_2))\\
u_{\mathrm{F},\mathrm{G}}(f^{out}_{\mathrm{G}}(T_4)) &\to u_{\mathrm{G},\mathrm{H}}(f^{in}_{\mathrm{A}}(T_1, T_4), T_4)\\
u_{\mathrm{G},\mathrm{H}}(f^{out}_{\mathrm{A}}, T_4) &\to f^{out}_{\mathrm{F}}(T_4)
\end{aligned}
$$



intuition:

$f^{in}_{\mathrm{F}}(T_1, T_2)$ evaluates to $f^{out}_{\mathrm{F}}(T_4)$ if $T_1 \neq [\,]$, $T_2 \neq [\,]$, T_4 is T_2 without prefix T_1, $T_4 \in (T_1)^*$

$f^{in}_{\mathrm{G}}(T_1, T_2)$ evaluates to $f^{out}_{\mathrm{G}}(T_4)$ if $T_1 \neq [\,]$, $T_2 \neq [\,]$, T_4 is T_2 without prefix T_1

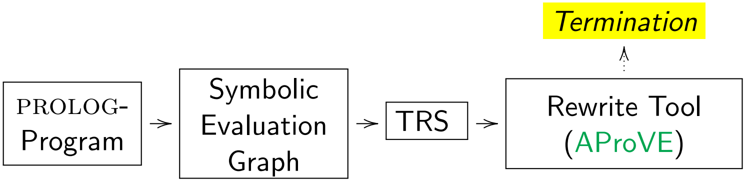
$f^{in}_{\mathrm{A}}(T_1, T_4)$ evaluates to f^{out}_{A} if $T_4 \in (T_1)^*$

$$
\begin{aligned}
\mathsf{star}(XS, []) &\text{ :- } !.\\
\mathsf{star}([], ZS) &\text{ :- } !, \mathsf{eq}(ZS, []).\\
\mathsf{star}(XS, ZS) &\text{ :- } \mathsf{app}(XS, YS, ZS), \mathsf{star}(XS, YS).\\
\mathsf{app}([], YS, YS)&.\\
\mathsf{app}([X \mid XS], YS, [X \mid ZS]) &\text{ :- } \mathsf{app}(XS, YS, ZS).\\
\mathsf{eq}(X, X)&.
\end{aligned}
$$

$$
\begin{aligned}
f^{in}_{\mathrm{A}}(T_1, T_2) &\rightarrow \mathsf{u}_{\mathrm{A,F}}(f^{in}_{\mathrm{F}}(T_1, T_2))\\
\mathsf{u}_{\mathrm{A,F}}(f^{out}_{\mathrm{F}}(T_3)) &\rightarrow f^{out}_{\mathrm{A}}\\
f^{in}_{\mathrm{A}}(T_1, []) &\rightarrow f^{out}_{\mathrm{A}}\\
f^{in}_{\mathrm{F}}(T_1, T_2) &\rightarrow \mathsf{u}_{\mathrm{F,G}}(f^{in}_{\mathrm{G}}(T_1, T_2))\\
\mathsf{u}_{\mathrm{F,G}}(f^{out}_{\mathrm{G}}(T_4)) &\rightarrow \mathsf{u}_{\mathrm{G,H}}(f^{in}_{\mathrm{A}}(T_1, T_4), T_4)\\
\mathsf{u}_{\mathrm{G,H}}(f^{out}_{\mathrm{A}}, T_4) &\rightarrow f^{out}_{\mathrm{F}}(T_4)\\
f^{in}_{\mathrm{G}}([T_5 \mid T_6], [T_5 \mid T_7]) &\rightarrow \mathsf{u}_{\mathrm{G,I}}(f^{in}_{\mathrm{I}}(T_6, T_7))\\
\mathsf{u}_{\mathrm{G,I}}(f^{out}_{\mathrm{I}}(T_3)) &\rightarrow f^{out}_{\mathrm{G}}(T_3)\\
f^{in}_{\mathrm{I}}([T_8 \mid T_9], [T_8 \mid T_{10}]) &\rightarrow \mathsf{u}_{\mathrm{I,K}}(f^{in}_{\mathrm{I}}(T_9, T_{10}))\\
\mathsf{u}_{\mathrm{I,K}}(f^{out}_{\mathrm{I}}(T_3)) &\rightarrow f^{out}_{\mathrm{I}}(T_3)\\
f^{in}_{\mathrm{I}}([], T_3) &\rightarrow f^{out}_{\mathrm{I}}(T_3)
\end{aligned}
$$

- existing TRS tools prove termination automatically
- original PROLOG program terminates

Symbolic Evaluation Graphs and Term Rewriting

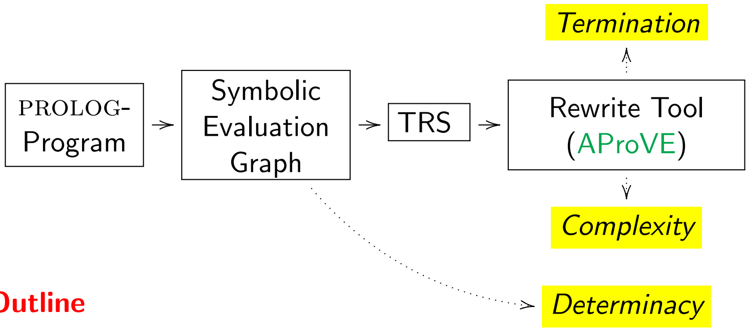


implemented in tool **AProVE**

- most powerful tool for termination of definite logic programs
- only tool for termination of non-definite PROLOG programs
- winner of *termination competition* for PROLOG
  (proves 342 of 477 examples, average runtime 6.5 s per example)

Symbolic Evaluation Graphs and Term Rewriting

General methodology for analyzing PROLOG programs



Outline

- linear operational semantics of PROLOG
- from PROLOG to symbolic evaluation graphs
- from symbolic evaluation graphs to TRSs for termination analysis
- from symbolic evaluation graphs to TRSs for complexity analysis
- determinacy analysis

Complexity for TRSs

$$\mathcal{R}: \quad \begin{array}{rcl} \mathsf{double}(0) & \to & 0 \\ \mathsf{double}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \end{array}$$

- $irc_{\mathcal{R}}$ maps $n \in \mathbb{N}$ to maximal evaluation starting with basic term t, where $|t| \leq n$
- $|t|$: number of variables and function symbols in t

$$\begin{array}{lll} \mathsf{double}(\mathsf{s}^k(0)) & \to_{\mathcal{R}}^{k+1} & \mathsf{s}^{2 \cdot k}(0) \\ \mathsf{double}^k(\mathsf{s}(0)) & \to_{\mathcal{R}}^{2^k+k-1} & \mathsf{s}^{2^k}(0) \end{array}$$

- only consider **basic terms** $f(t_1, \ldots, t_n)$

  f defined symbol (double), $t_1, \ldots, t_n$ no defined symbols (s, 0)

Complexity for TRSs

$$\mathcal{R}: \quad \begin{array}{rcl} \mathsf{double}(0) & \to & 0 \\ \mathsf{double}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \end{array}$$

- $irc_{\mathcal{R}}$ maps $n \in \mathbb{N}$ to maximal evaluation starting with basic term t, where $|t| \leq n$
- $|t|$: number of variables and function symbols in t

$$\mathsf{double}(\mathsf{s}^k(0)) \quad \to_{\mathcal{R}}^{k+1} \quad \mathsf{s}^{2\cdot k}(0)$$

- $\mathcal{R}$ has linear complexity if $irc_{\mathcal{R}}(n) \in \mathcal{O}(n)$
  $\mathcal{R}$ has quadratic complexity if $irc_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$ etc.
- Example: has linear complexity
- Recently: many powerful techniques for complexity of TRSs
  (by adapting techniques for termination analysis)

Complexity for Logic Programs

Program $\mathcal{P}$, Class of queries $\mathcal{Q}^{\mathsf{p}}_m$

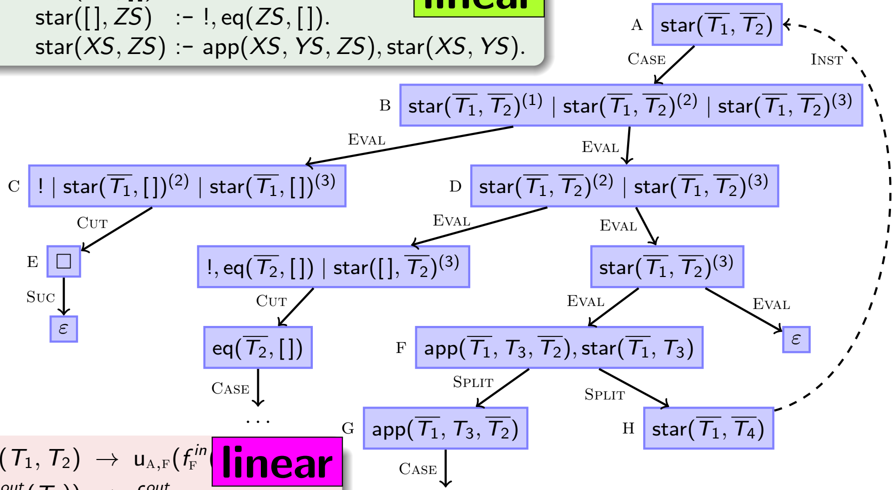
- $prc_{\mathcal{P},\mathcal{Q}^{\mathsf{p}}_m}$ maps $n \in \mathbb{N}$ to longest evaluation starting with $Q \in \mathcal{Q}^{\mathsf{p}}_m$, where $|Q|_m \leq n$
- $|Q|_m$: number of variables and function symbols on *input positions*
- corresponds to number of unification attempts

- $\mathcal{P}$ has linear complexity for class $\mathcal{Q}^{\mathsf{p}}_m$ if $prc_{\mathcal{P},\mathcal{Q}^{\mathsf{p}}_m}(n) \in \mathcal{O}(n)$
  $\mathcal{P}$ has quadratic complexity for class $\mathcal{Q}^{\mathsf{p}}_m$ if $prc_{\mathcal{P},\mathcal{Q}^{\mathsf{p}}_m}(n) \in \mathcal{O}(n^2)$ etc.
- Example (star-program): has linear complexity
- Goal: Re-use existing methodology for termination analysis
  to analyze complexity as well

$\mathcal{P}$:

```
star(XS, [])  :- !.
star([], ZS)  :- !, eq(ZS, []).
star(XS, ZS)  :- app(XS, YS, ZS), star(XS, YS).
```

linear



$$
\begin{aligned}
f^{in}_{\mathrm{A}}(T_1, T_2) &\rightarrow \mathsf{u}_{\mathrm{A,F}}(f^{in}_{\mathrm{F}}(\\
\mathsf{u}_{\mathrm{A,F}}(f^{out}_{\mathrm{F}}(T_3)) &\rightarrow f^{out}_{\mathrm{A}} \\
f^{in}_{\mathrm{A}}(T_1, [\,]) &\rightarrow f^{out}_{\mathrm{A}} \\
f^{in}_{\mathrm{F}}(T_1, T_2) &\rightarrow \mathsf{u}_{\mathrm{F,G}}(f^{in}_{\mathrm{G}}(T_1, T_2)) \\
\mathsf{u}_{\mathrm{F,G}}(f^{out}_{\mathrm{G}}(T_4)) &\rightarrow \mathsf{u}_{\mathrm{G,H}}(f^{in}_{\mathrm{A}}(T_1, T_4), T_4) \\
\mathsf{u}_{\mathrm{G,H}}(f^{out}_{\mathrm{A}}, T_4) &\rightarrow f^{out}_{\mathrm{F}}(T_4)
\end{aligned}
$$

linear

- generate symbolic evaluation graph
- generate TRS from graph
- determine complexity of TRS by existing tool
- infer that $\mathcal{P}$ has the same complexity

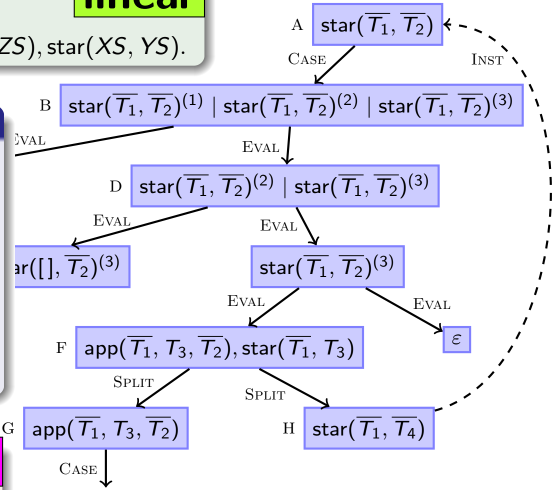
$\mathcal{P}$:

```
star(XS, [])  :- !.
star([], ZS)  :- !, eq(ZS, []).
star(XS, ZS)  :- app(XS, YS, ZS), star(XS, YS).
```

linear

Correct?

- depends on SPLIT's successor G
- in $\mathcal{P}$: repeat evaluation of H for every answer of G (*backtracking*)
- in TRS: evaluate H once (choose G's answer *non-deterministically*)
- Here: G is *deterministic* (has only one answer)



$$
\begin{aligned}
f^{in}_{A}(T_1, T_2) &\rightarrow u_{A,F}(f^{in}_{F} \\
u_{A,F}(f^{out}_{F}(T_3)) &\rightarrow f^{out}_{A} \\
f^{in}_{A}(T_1, []) &\rightarrow f^{out}_{A} \\
f^{in}_{F}(T_1, T_2) &\rightarrow u_{F,G}(f^{in}_{G}(T_1, T_2)) \\
u_{F,G}(f^{out}_{G}(T_4)) &\rightarrow u_{G,H}(f^{in}_{A}(T_1, T_4), T_4) \\
u_{G,H}(f^{out}_{A}, T_4) &\rightarrow f^{out}_{F}(T_4)
\end{aligned}
$$

linear

- generate symbolic evaluation graph
- generate TRS from graph
- determine complexity of TRS by existing tool
- infer that $\mathcal{P}$ has the same complexity

$\mathcal{P}:$ $\text{sublist}(X, Y)$:- $\text{app}(P, U, Y)$,

quadratic

$\text{app}([\,], YS, YS).$

$\text{app}([X \mid XS], YS, [X \mid ZS])$:- $\text{app}(XS, YS, ZS)$ (3)

Evaluation of sublist

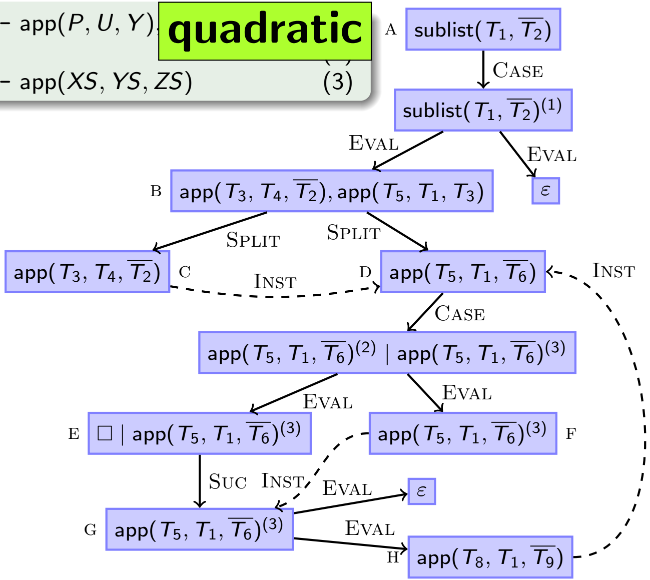
- $\mathcal{Q}_m^{\text{sublist}} = \{\text{sublist}(t_1, t_2) \mid t_2 \text{ ground}\}$
- computes all sublists of Y (by *backtracking*)
- $\mathcal{P}$: quadratic complexity
  - linear many possibilities to split Y into P and U
  - for each possible P, linear evaluation of $\text{app}(V, X, P)$

$\mathcal{P}$: sublist(X, Y) :- app(P, U, Y),

quadratic

app([ ], YS, YS).

app([X | XS], YS, [X | ZS]) :- app(XS, YS, ZS) (3)



$f^{in}_{\mathrm{B}}(T_2) \to$ u

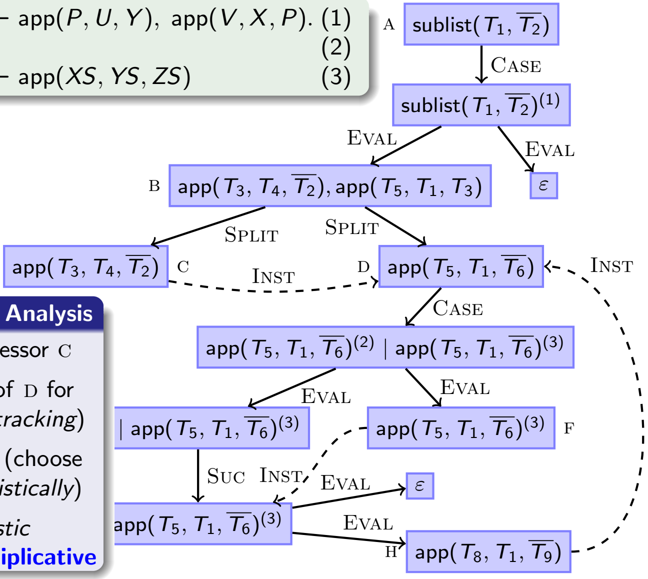
linear

$u_{\mathrm{B,C}}(f^{out}_{\mathrm{D}}(\ldots)) \to u_{\mathrm{C,D}}(f^{in}_{\mathrm{D}}(\ldots))$

$u_{\mathrm{C,D}}(f^{out}_{\mathrm{D}}(\ldots)) \to f^{out}_{\mathrm{B}}(\ldots)$

- generate symbolic evaluation graph and TRS
- determine complexity of TRS by existing tool
- ~~infer that $\mathcal{P}$ has the same complexity~~

$$
\begin{array}{lrr}
\mathcal{P}: & \mathsf{sublist}(X, Y) \text{ :- } \mathsf{app}(P, U, Y),\ \mathsf{app}(V, X, P). & (1) \\
& \mathsf{app}([\,], YS, YS). & (2) \\
\multicolumn{2}{l}{\mathsf{app}([X \,|\, XS], YS, [X \,|\, ZS]) \text{ :- } \mathsf{app}(XS, YS, ZS)} & (3)
\end{array}
$$



Correctness of Complexity Analysis

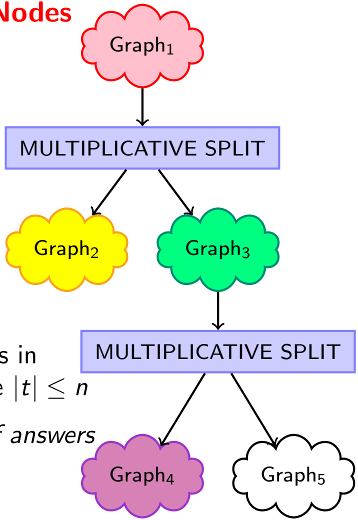
- depends on SPLIT's successor C
- in $\mathcal{P}$: repeat evaluation of D for every answer of C (*backtracking*)
- in TRS: evaluate D once (choose C's answer *non-deterministically*)
- Here: C is *not deterministic* $\Rightarrow$ SPLIT node B is **multiplicative**

$$
\begin{array}{rcl}
f_{\mathrm{B}}^{in}(T_2) & \to & \mathsf{u}_{\mathrm{B,C}}(f_{\mathrm{D}}^{in}(T_2)) \\
\mathsf{u}_{\mathrm{B,C}}(f_{\mathrm{D}}^{out}(\ldots)) & \to & \mathsf{u}_{\mathrm{C,D}}(f_{\mathrm{D}}^{in}(\ldots)) \\
\mathsf{u}_{\mathrm{C,D}}(f_{\mathrm{D}}^{out}(\ldots)) & \to & f_{\mathrm{B}}^{out}(\ldots)
\end{array}
$$

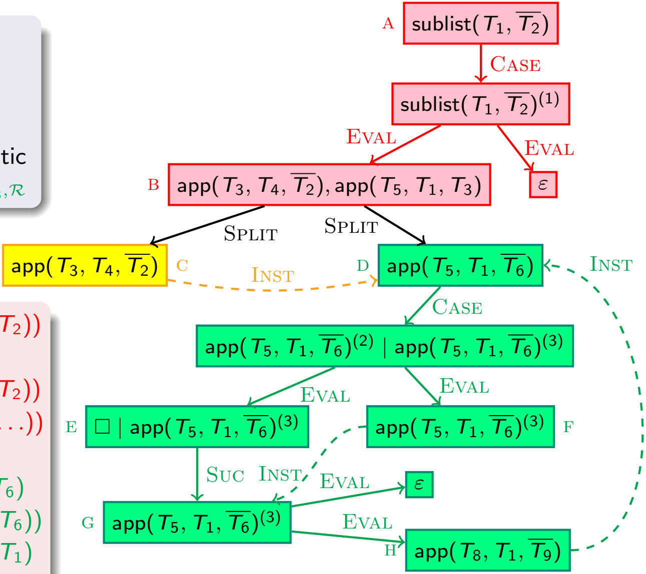
- generate symbolic evaluation graph and TRS
- determine complexity of TRS by existing tool
- ~~infer that $\mathcal{P}$ has the same complexity~~

Decompose Graph by Multiplicative Split Nodes

- generate symbolic evaluation graph
- generate separate TRSs $\mathcal{R}_1, \ldots, \mathcal{R}_5$ for parts up to multiplicative Split nodes (no multiplicative Split node may reach itself)
- determine $irc_{\mathcal{R}_1,\mathcal{R}}$, ..., $irc_{\mathcal{R}_5,\mathcal{R}}$ separately
  - maps $n \in \mathbb{N}$ to maximal number of $\mathcal{R}_i$-steps in evaluation starting with basic term t, where $|t| \leq n$
  - upper bound for *runtime* and for *number of answers*
- combine complexities
  - multiply complexities for children of multiplicative Splits
  - add complexities of parents of multiplicative Splits
  - $irc_{\mathcal{R}_1,\mathcal{R}} + irc_{\mathcal{R}_2,\mathcal{R}} \cdot (irc_{\mathcal{R}_3,\mathcal{R}} + irc_{\mathcal{R}_4,\mathcal{R}} \cdot irc_{\mathcal{R}_5,\mathcal{R}})$



- $irc_{\mathcal{R}_1,\mathcal{R}}$: constant
- $irc_{\mathcal{R}_2,\mathcal{R}}$: linear
- $irc_{\mathcal{R}_3,\mathcal{R}}$: linear
- complexity of $\mathcal{P}$: quadratic
  $irc_{\mathcal{R}_1,\mathcal{R}} + irc_{\mathcal{R}_2,\mathcal{R}} \cdot irc_{\mathcal{R}_3,\mathcal{R}}$



$$
\begin{array}{rcl}
f^{in}_{A}(T_2) & \to & u_{A,B}(f^{in}_{B}(T_2)) \\
u_{A,B}(f^{out}_{B}(\ldots)) & \to & f^{out}_{A}(T_1) \\
f^{in}_{B}(T_2) & \to & u_{B,C}(f^{in}_{D}(T_2)) \\
u_{B,C}(f^{out}_{D}(\ldots)) & \to & u_{C,D}(f^{in}_{D}(\ldots)) \\
u_{C,D}(f^{out}_{D}(\ldots)) & \to & f^{out}_{B}(\ldots) \\
f^{in}_{D}(T_6) & \to & f^{out}_{D}([\,], T_6) \\
f^{in}_{D}(T_6) & \to & u_{D,G}(f^{in}_{G}(T_6)) \\
u_{D,G}(f^{out}_{G}(\ldots)) & \to & f^{out}_{D}(T_5, T_1) \\
f^{in}_{D}(T_6) & \to & u_{D,F}(f^{in}_{G}(T_6)) \\
u_{D,F}(\ldots) & \to & f^{out}_{D}(T_5, T_1) \\
f^{in}_{G}([T_7 \mid T_9]) & \to & u_{G,H}(\ldots) \\
u_{G,H}(\ldots) & \to & f^{out}_{G}([T_7 \mid T_8])
\end{array}
$$

- generate graph and TRSs $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$
- determine $irc_{\mathcal{R}_1,\mathcal{R}}$, $irc_{\mathcal{R}_2,\mathcal{R}}$, $irc_{\mathcal{R}_3,\mathcal{R}}$
- infer complexity of $\mathcal{P}$

Symbolic Evaluation Graphs and Term Rewriting

PROLOG-Program → Symbolic Evaluation Graph → TRS → Rewrite Tool (AProVE)

⇣

Complexity

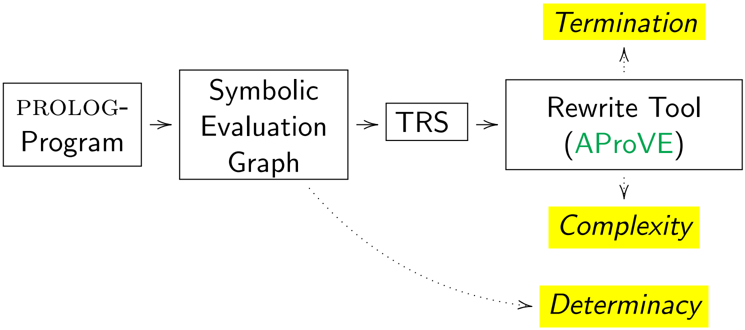
implemented in tool **AProVE**

- only tool for complexity of non-well-moded or non-definite programs
- experiments on all 477 programs of *TPDB*

	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot 2^n)$	**bounds**	**time**
CASLOG	1	21	4	**3**	29	14.8
CiaoPP	3	19	4	**3**	29	11.7
AProVE	**54**	**117**	**37**	0	**208**	**10.6**

Symbolic Evaluation Graphs and Term Rewriting

General methodology for analyzing PROLOG programs



Outline

- linear operational semantics of PROLOG
- from PROLOG to symbolic evaluation graphs
- from symbolic evaluation graphs to TRSs for termination analysis
- from symbolic evaluation graphs to TRSs for complexity analysis
- determinacy analysis

Criterion for determinacy of s

If s reaches SUC node s', then there is no path from s' to a SUC node.

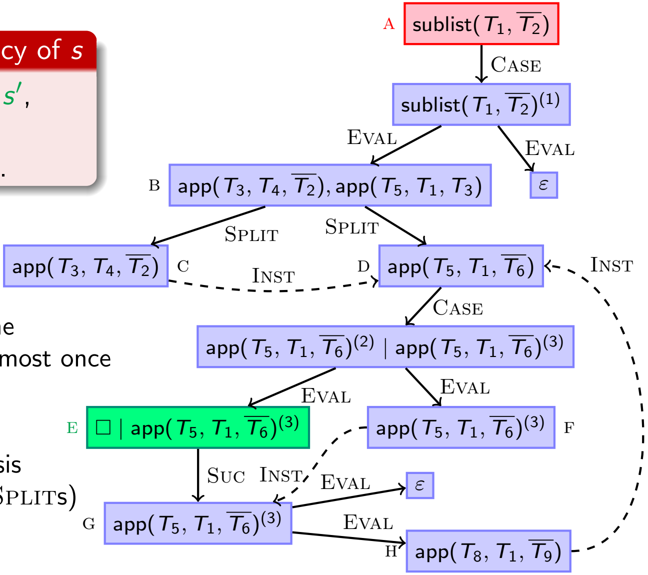
- query deterministic iff

  it generates at most one answer substitution at most once

  - for program analysis
  - for complexity analysis (non-multiplicative SPLITs)

- successful evaluation ⇒

  path to SUC node in symbolic evaluation graph

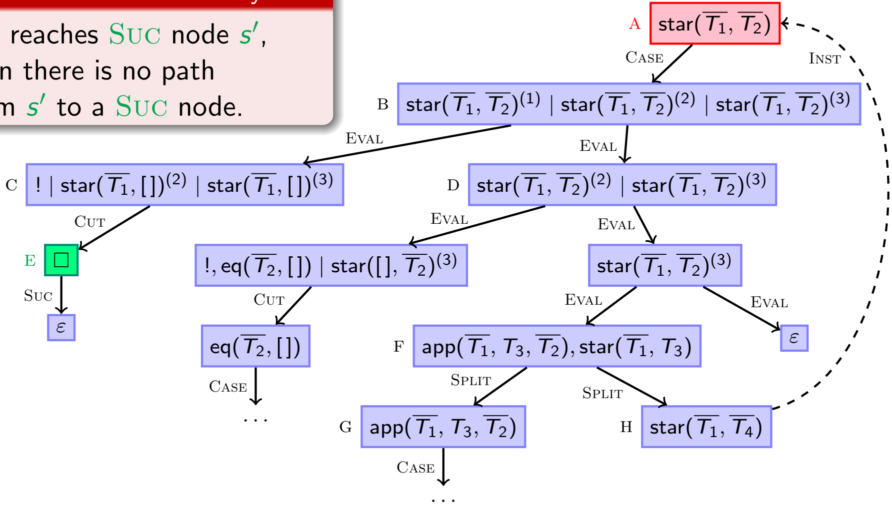


- C not deterministic

  ⇒ SPLIT node B multiplicative
- A not deterministic

Criterion for determinacy of s

If s reaches SUC node s', then there is no path from s' to a SUC node.



- G is deterministic
  $\Rightarrow$ SPLIT node F not multiplicative
- A is deterministic

Symbolic Evaluation Graphs and Term Rewriting

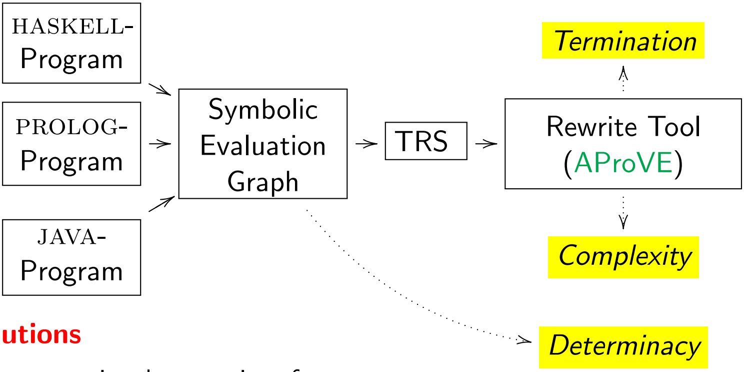
PROLOG-Program → Symbolic Evaluation Graph → *Determinacy*

implemented in tool **AProVE**

- experiments on 300 definite programs:
  CiaoPP: 132, AProVE: 80
- experiments on 177 non-definite programs:
  CiaoPP: 61, AProVE: 92
- only first step, but substantial addition to existing determinacy analyses
  (AProVE succeeds on 78 examples where CiaoPP fails)
- strong enough for complexity analysis

Symbolic Evaluation Graphs and Term Rewriting

General methodology for analyzing PROLOG programs



Contributions

- linear operational semantics of PROLOG
- from PROLOG to symbolic evaluation graphs
- from symbolic evaluation graphs to TRSs for termination analysis
- from symbolic evaluation graphs to TRSs for complexity analysis
- determinacy analysis

http://aprove.informatik.rwth-aachen.de