From Notation Theory to Expression Reduction Systems

Zurab Khasidashvili

Intel, Haifa

Sept 10 2019, Tbilisi, Georgia

Dedicated to Work of Shalva Pkhakadze on the Centenary of His Birth

Our goal in this talk is to describe contribution of the *Notation Theory* of Shalva Pkhakadze to the theory of Higher Order Rewrite Systems

Term Rewriting Systems -- an example

Rewrite rules for Peano arithmetic

- **A**(x,**0**) -> x
- **A**(x,**S**(y)) -> S(A(x,y))
- **M**(x,**0**) -> 0
- **M**(x,**S**(y)) -> A(M(x,y), x)

Axioms of Peano Arithmetic

- x+0 = x
- x+(y+1) = (x+y)+1
- x*0 = 0
- x*(y+1) = x*y+x

- Numbers: 0, S(0), S(S(0)), …
- Examples of *reductions* (contraction of *redexes*)
  - **M(**S0,**S**0) -> A(**M(**S0,**0**),S0) -> **A**(0,**S**0) -> S**A(**0,**0)** -> S0
  - **M(**S0,**S**0) -> **A(**M(S0,0),**S**0) -> SA(**M(**S0,**0**),0) -> S**A**(0,**0**) -> S0
  - S0 is a *normal form* (contains no redexes, no further reduction is possible)

Lambda Calculus

- Formalizes computation; has the same computational power as Turing Machines
- Expressions (terms) are built using two operations:
  - Abstraction: λx; example: λx.M
  - Application: (MN), which abbreviates @(M,N)
- Beta rule: (λx.M)N -> (M/x)N
  @(λx.M, N) -> (M/x)N
- Substitution
  - (λx.x)N -> N
  - (λx.y)N -> y
  - (λx.(λx.xy)) N -> λx.xy
  - (λz.(λx.xy)) N -> λx.xy – *renaming of bound variables*

Quantifiers in First Order Logic

- Existential quantifier: $\exists x.\varphi(x)$
- Universal quantifier: $\forall x.\varphi(x)$

- Expressions are considered as same up to renaming of bound variables:
  - $\exists x.\varphi(x) \equiv \exists y.\varphi(y)$

- Defined symbols of the form $\sigma\ a1 \ldots an\ (A1,\ldots.,Am)$ -- B
  - Hilbert's choice operator τ: $\exists aA$ --- $(\tau x(A)/a)\ A$
  - Exits Unique operator $\exists!$: $\exists!aA$ ---- $\exists aA \wedge \forall a \forall b(A \wedge (b/a)A \rightarrow a = b)$,

Definition of symbols with binding power

- Need to define rigorous rules to make definition of quantifiers sound – avoid any collision between free and bound variables
- Such a definition will allow meta-level reasoning on properties of such symbols when considered as rewrite rules
- For example, Boubaki in *Elements of Mathematics*, Chapters 1 & 2, introduced “defined symbols” without a formal definition of what a “definition” of a new symbol (a quantifier or a function) is; thus they couldn’t prove any general properties of defined symbols and extensions of theories with defined symbols. The Notation Theory was introduced by Sh. Pkhakadze to fix this situation

Definition of contracting symbols of type IV

- Let a1,...,am be metavariables such that each ai ranges over the class of all predicate or object quantifier variables or letters; and let A1,...,An be metavariables such that each Aj ranges over the class of all formulas or all terms. The definition of a contracted symbol σ of type IV has a form

  σa1 ...amA1 ...An --- B

  where a1,...,am,A1,...,An are metavariables, each ranged over a class of quantifier letters or forms (as specified above), and B is a form constructed using the main and already introduced contracting symbols, metavariables a1,...,am, b1,...bk, and (/)-substitutions. A system of values a'1...a'm,A'1,...A'n,b'1,...,b'k is admissible if b'1,...,b'k are mutually different, are different from a'1,...a'm, do not have free occurrences in A'1,...A'n, and do not have (any) occurrences in B.

  - Example (exits unique): ∃!aA ---- ∃aA∧∀a∀b(A∧(b/a)A→a = b),

Definition of contracting symbols of types I-III

- If we require that B contains no (/)-substitutions, then from definitions of types IV we get definitions of type II
  - Example (subset operator): $\subseteq AB$ ---- $\forall b(b \in A \rightarrow b \in B)$.
- And if we require that the list of additional metavariables b1,...,bk be empty, we get definitions of types I and III from the definitions of types II and IV, respectively.
  - There are also contracting symbols of types IV' and II'; their definitions are slight adaptations (and do not change the expressive power) of definitions of types IV and II, respectively; and there are also definitions of a few other types designed to illustrate that relaxing constraints on definitions of types I-IV would result in loss of many desirable properties of contracted symbols

Some work on theory of contracting symbols and extensions

- Khimur Rukhaia -- The description of a derived formal mathematical T^* theory, 1983
- Vakhtang Pkhakadze -- Some properties of α-processes, 1988
- Vakhtang Pkhakadze -- Substitution theorems and connection between δ-processes and α-processes, 1988
- Zurab Khasidashvili – Expression Reduction Systems, 1990

How are Expression Reduction Systems (ERSs) Different?

- The essential difference is in the left-hand sides of definitions (viewed as rules): In ERSs, the LHS may be expressions just like the RHS expression B except they cannot contain substitution operation (/)
- This generality in the LHS allows more computations to be expressed
  - The λ–Calculus and Term Rewriting Systems are a special case

Some History of rewrite systems with bound variables and substitution (with no aim or claim of completeness….)

- Main motivating examples/theories
  - Thirst Order Logic
  - λ–calculus – Alonso Church, The Calculi of Lambda Conversion, 1941

- The earliest formalizms (with partial power for rewriting)
  - **Shalva Pkhakadze – 1977, Some Problems of the Notation Theory**
  - Peter Aczel – 1978, a General Church-Rosser Theorem

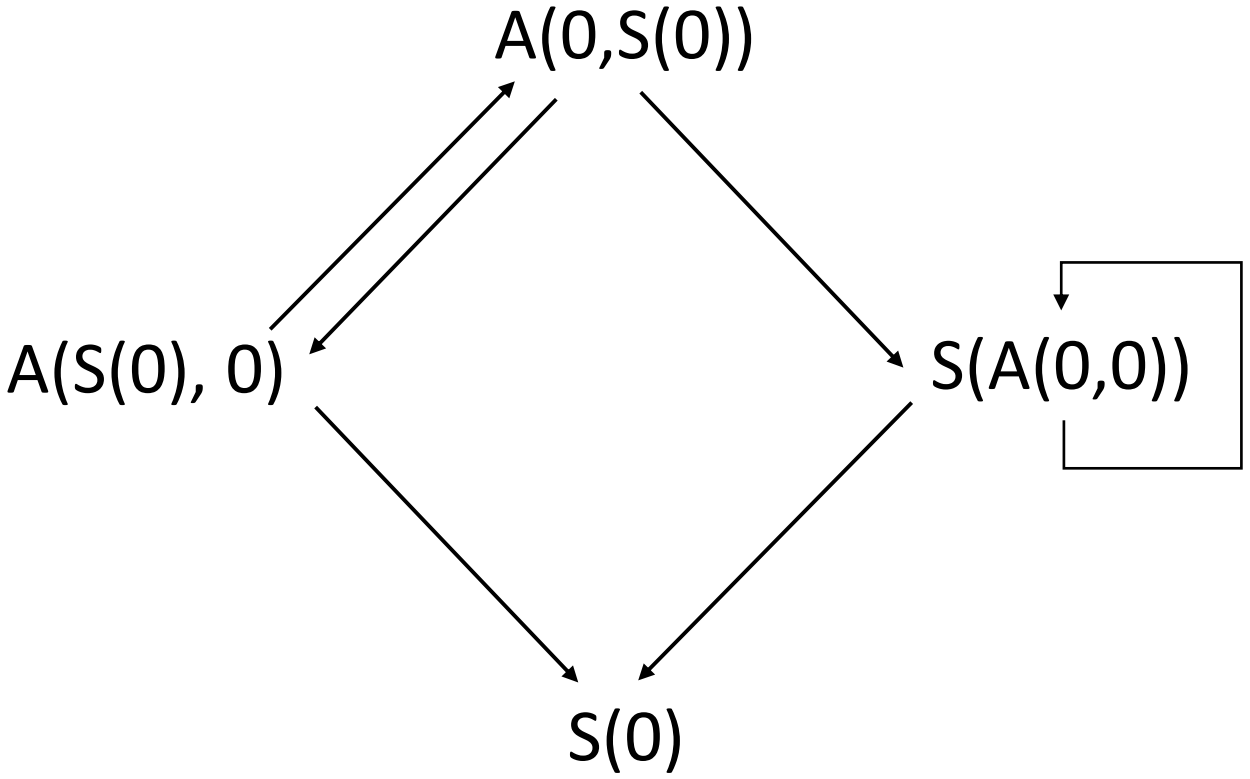
- First full formalization, with proofs of many basic theorems on confluence and normalization
  - Jan Willem Klop – 1980, Combinatory Reduction Systems, PhD thesis

Some other well known formalizms

- Other popular formalizms
  - Zurab Khasidashvili – Expression Reduction Systems, 1990
  - Tobias Nipkow -- High-Order Rewrite Systems – Tobias Nipkow, 1991
  - David Wolfram – Higher-Order Term Rewriting Systems, 1993
  - Vincent Van Oostrom and Femke van Raamsdonk – Higher-Order Rewrite Systems, 1994
  - Julien Forest and Delia Kesner -- Expression reduction systems with patterns, 2003
  - ....
  - Many more formalizms exist for *combining Term Rewriting and* λ*-Calculus*
  - A big body of related research is on *Explicit Substitution* aiming and defining / implementing substitution as Term Rewriting

Some of the Basic Questions for Rewrite Systems

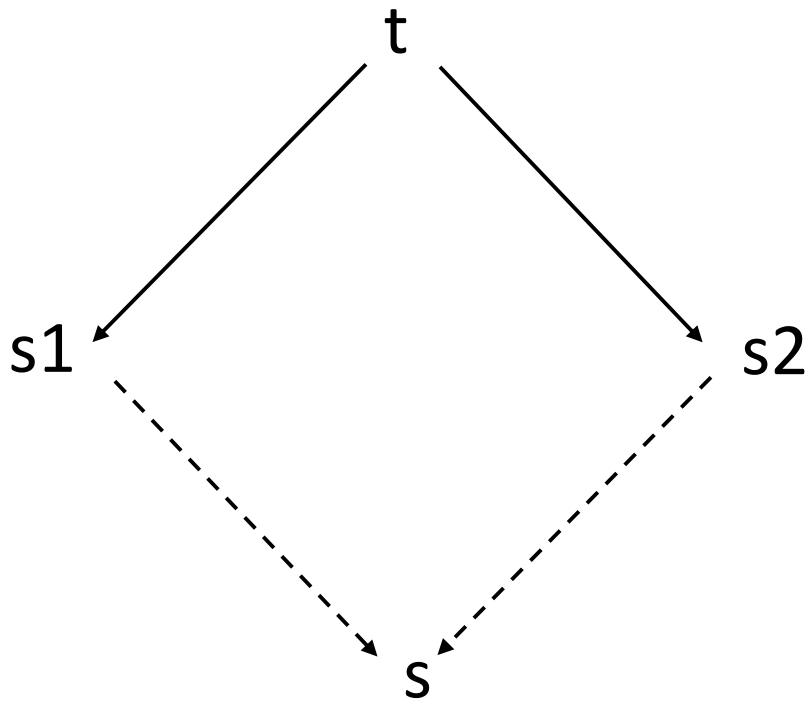
- Confluence
- Weak and strong normalization
- Normalizing and Perpetual strategies
- Optimal Reduction



Peano Arithmetic with commutativity rule for addition:
A(x,y) -> A(y,x)

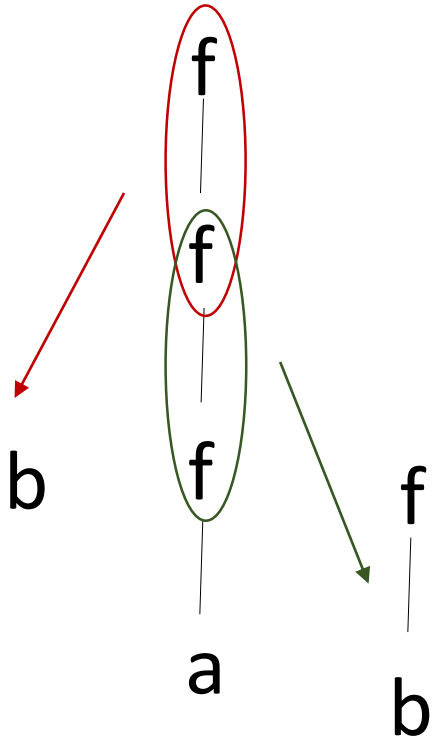
Confluence

- Confluence (or Church-Rosser): Given two computations t ->> s1 and t ->> s2 from the same expression t, there exist reductions s1 ->> s and s2 ->> s to the same expression s
- Confluence implies uniqueness of a normal form for t (final result of computation of t) if a normal form exists



Orthogonal rewrite systems

- Left-liner systems where **redex patterns** do not overlap are called *orthogonal*.
- Orthogonal rewrite systems are confluent
- Example: Consider rule
  f(f(x)) -> b
  and consider expression
  t = f(f(f(a)))
  There are two redexes in t and their patterns overlap (form a critical pair)



Weak and Strong Normalization

- Weak Normalization: does an expression t have a normal form?
- Strong Normalization: Do all reductions (computations) of a term t terminate?
- Weak and strong normalization are defined for the Rewrite Systems – is any expression on the system weakly/strongly normalizing?

- Let w = λx.(xx)λx(xx) and v = (λy. z)w. Then:
- w has an infinite reduction w -> w -> w -> …..
- Thus v has an infinite reduction contracting the subexpression w within v: v -> v -> v ->….
- W foes not have a normal form but v does: v = (λy. z)w -> z
  - W is **erased** because variable y does not occur in expression z

Normalizing and Perpetual Strategies

- For expressions that have a normal form and also have an infinite reduction (computation):
  - Normalizing strategies: is there a way to choose the order of computations that guarantees reaching a normal form?
  - Optimal reductions: How can we construct a normal form in least number of steps / with a minimal cost? *Sharing of redexes* or *Graph rewriting* is often used.
  - Perpetual strategies: is there a way to choose the order of computations that guarantees construction of an infinite reduction?
- An example normalizing strategy in orthogonal systems:
  - *Parallel outermost* strategy is normalizing: The strategy that contracts all outermost redexes in a term is normalizing
  - *Outermost-fair strategy* is normalizing: every outermost redex must eventually be contracted.
  - For Rewrite Systems where the rules have the form of contracting symbols of types I-IV, *innermost needed reductions* are the shortest and can be computed effectively

Thank You