Data-Flow-Based Normalization Generation Algorithm of R1CS for Zero-Knowledge Proof

Chenhao Shi, Hao Chen, Ruibang Liu, Guoqiang Li†

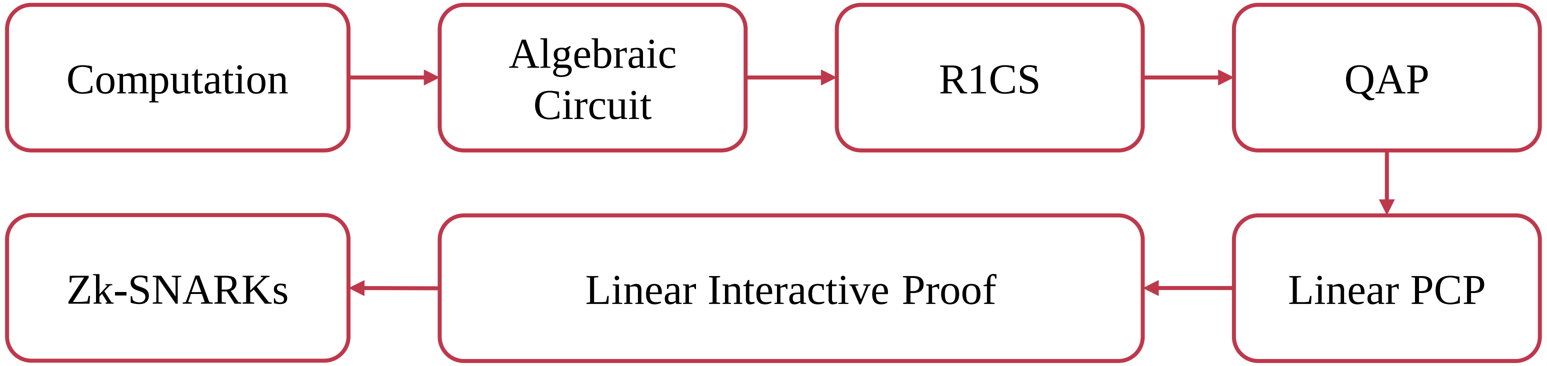
Shanghai Jiao Tong University

Contents

RESEARCH BACKGROUND

Research Background

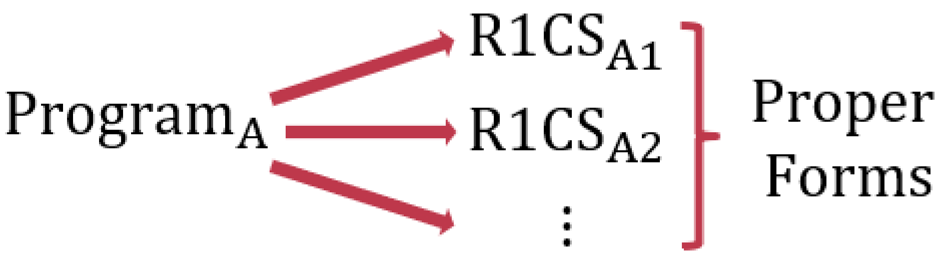
- **Increasing Importance**
  - Primary Applications: ***Privacy Security, Scalability***
  - Key Application Domains: ***Web3, Cryptocurrency, Financial Industry***
- **Complex Transformation Process**



Research Background

- **Limitations in Conversing Arithmetic Circuits to R1CS**
  - Limited Mergeability

  - Flexible R1CS Representations



ALGORITHM DESIGN

Algorithm Design

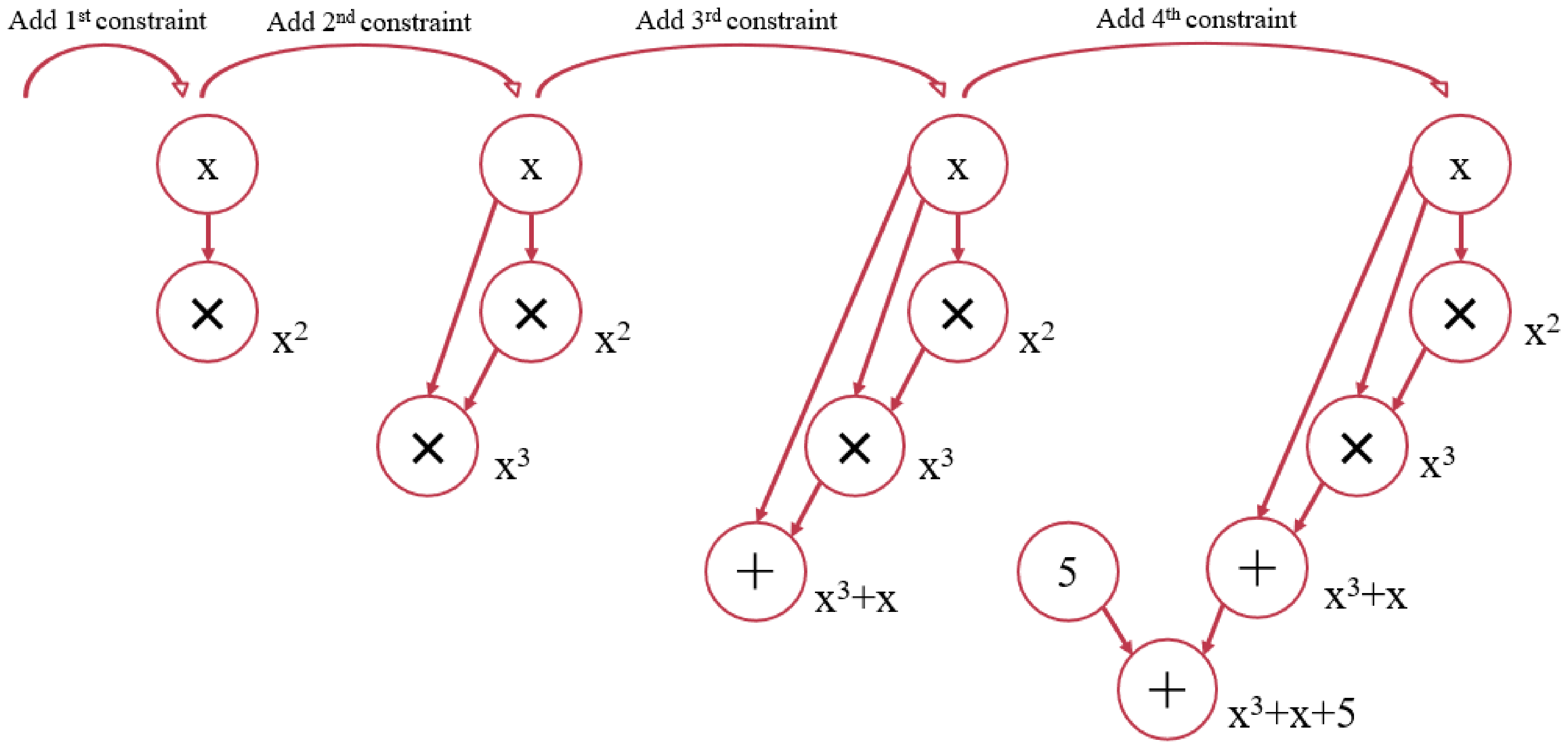
- **Example Equivalent R1CS**
  - $A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \mathrm{B} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \mathrm{C} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$

    $\text{Variable mapping} = (\sim\text{one}, \text{x}, , \sim\text{out}, \text{x}^2, \text{x}^3, \text{sym}_1)$
  - $A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ -5 & -1 & 1 & 0 \end{pmatrix}$

    $\text{Variable mapping} = (\sim\text{one}, \text{x}, , \sim\text{out}, \text{x}^2\ )$

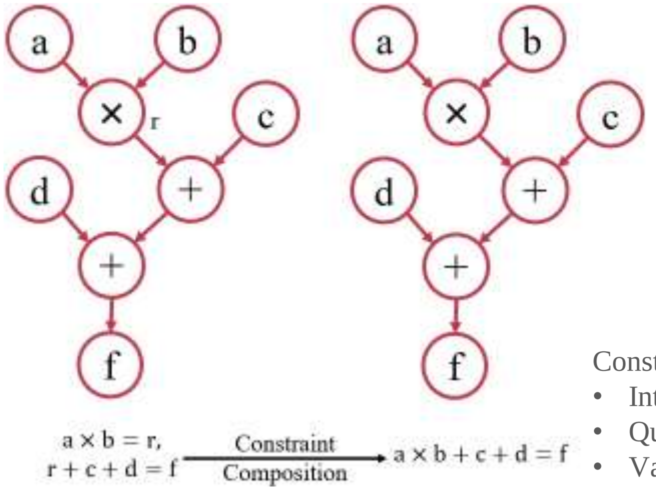
Algorithm Design: Construction of RNode Graph

- **Main Steps:** *Transform each equation and combine the results*



Algorithm Design: Construction of RNode Graph

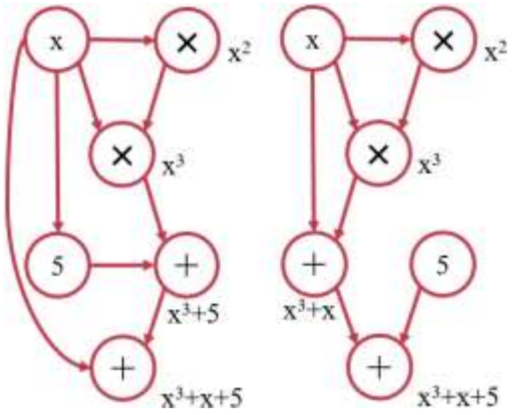
The design of the RNode data structure minimizes changes to the overall structure.



Constraint composition will affect:

- Intermediate variable selection.
- Quantity and form of constraints.
- Variable mapping.

Structural differences of RNode Graph of different R1CS remain.



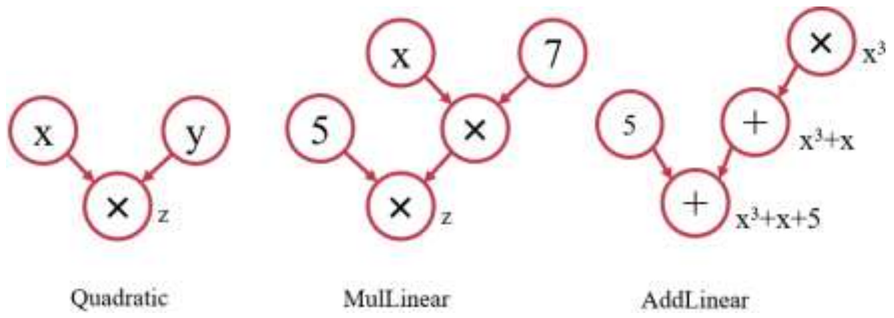
Reason: Consecutive additions

- Left: $(x^3 + x) + 5 = \text{output}$
- Right: $(x^3 + 5) + x = \text{output}$

Additional abstraction is necessary.

Algorithm Design: Tile Selection

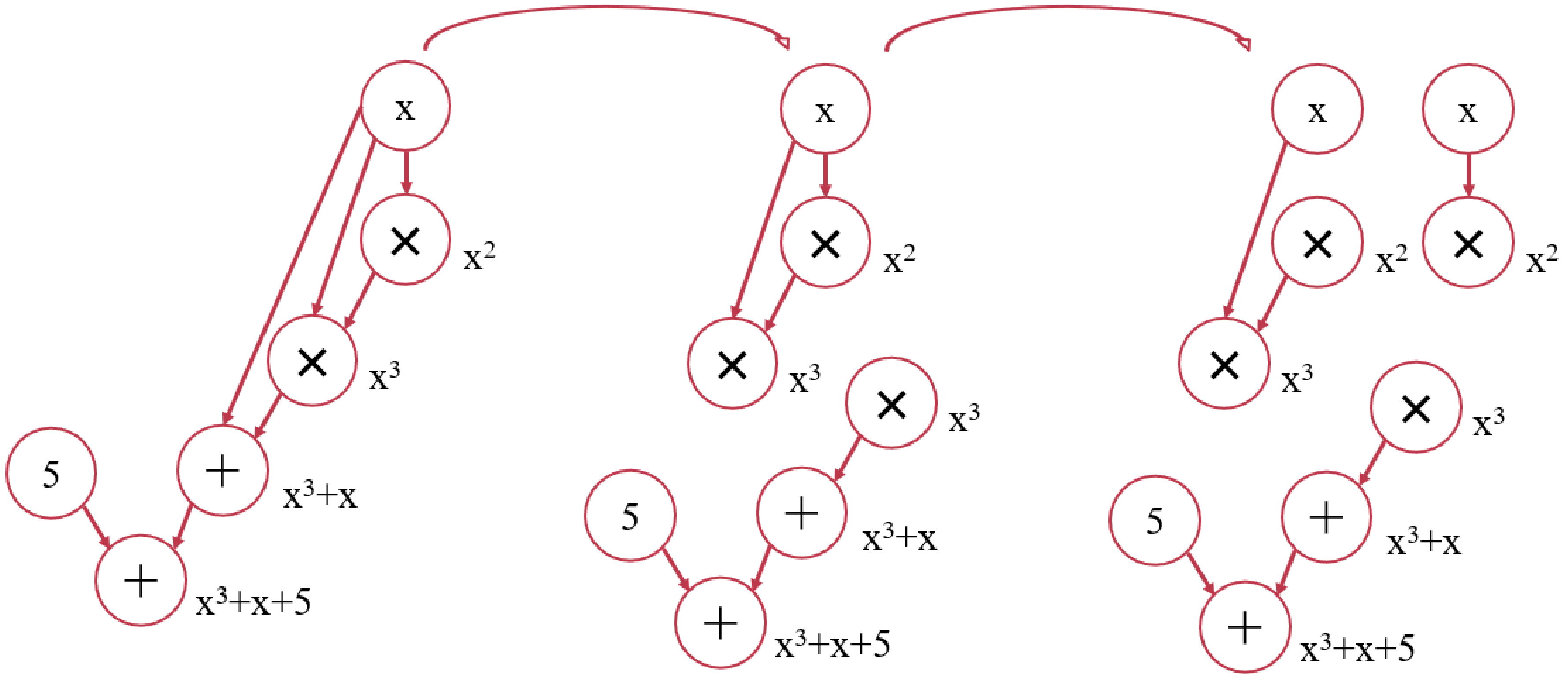
- **Main Steps:** ***Partitioning data flow graphs using customized tile types***
- **Customized Tile Types:**
  - Quadratic
  - MulLinear
  - AddLinear



- **Three Considerations:**
  - Temporarily deferring the constraint merging step.
  - Utilizing the paradigm based on unmerged constraints as the fundamental approach.
  - Implementing a relatively straightforward algorithm for tile selection.

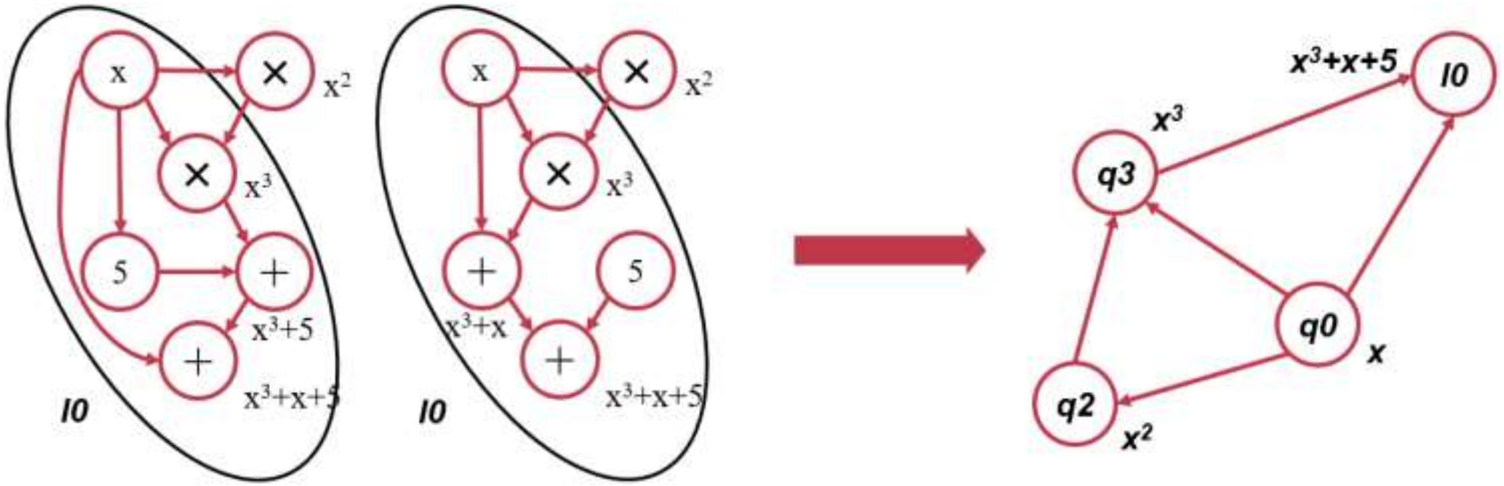
Algorithm Design: Tile Selection

- **The procedure of tile selection:**



Algorithm Design: RNode Graph Abstraction

- **Main Steps:** ***Abstracting linear tiles as a single large node and reconfiguring the associated edges.***



- **The type of edges after abstraction:**
  - Abstract Node to Abstract Node: ***Both abstracted nodes represent linear tiles that have common RNode.***
  - Abstract Node to RNode: ***RNode is present in the linear tile represented by the abstracted node.***
  - RNode to RNode: ***Maintaining consistency with the original data flow graph before abstraction.***

Algorithm Design: Tile Weight Calculation

- **Using Weighted PageRank Algorithm**
- **Formula for calculating weights:**
  - $PR(u) = (1-d) + d \sum_{v \in B(u)} PR(v) W_{(v,u)}^{in} W_{(v,u)}^{out}$
  - The weight $W_{(v,u)}^{in} W_{(v,u)}^{out}$ is computed using the in-degree and out-degree of node v and its neighboring nodes.
- **Using the variance of the coefficients as the weight of linear tiles.**
  - $W = \sum(a_i - \frac{\sum a_i}{n})^2 / (\frac{\sum a_i}{n})^2$
  - Diminishing the symmetry in the graph.
  - Removing identical weights of tiles.

Algorithm Design: Linear Constraint Adjustment

- **Scope of Adjustment :** ***Newly Introduced Variables in Linear Tiles.***
- **Sorting Criterion**
  - $weight = \sum_{other\ linear\ tiles} |\ field * weight\ of\ linear\ tile\ |$
- **Rationale:**
  - To reflect the significance of variables based on their occurrences within the constraints.
  - To maintain equal weights for variables that solely appear within a single linear constraint.

EVALUATION

Evaluation

- ## Benchmark Design
  - Benchmark Classification: ***According to summarized rules for generating equivalent R1CS***
  - Separated Generation: ***Within each category***

Developing benchmark constraint sets → Constructing equivalent constraint sets → Performing algorithm testing

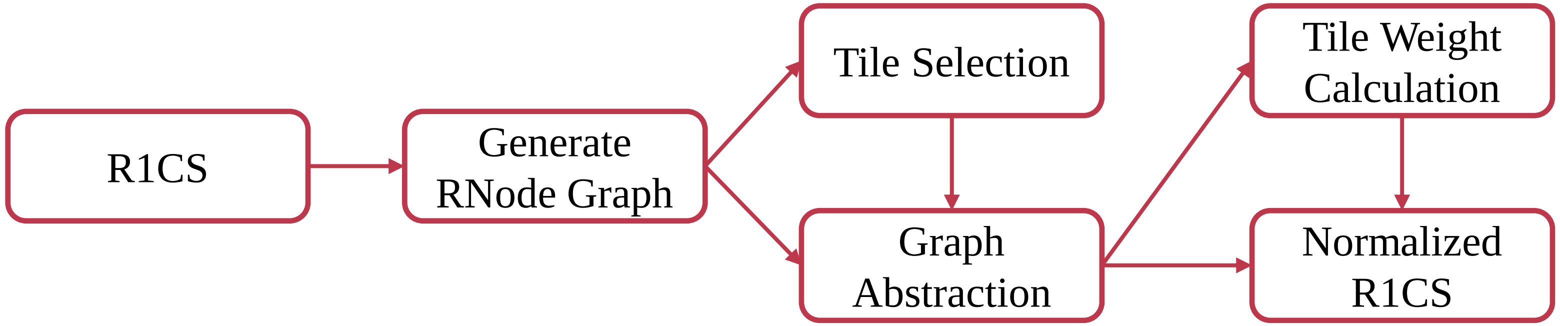
- ## Experiment Result Evaluation

Reasons for Generating Equivalent R1CS Constraints	*Number of Groups*	*Successfully Generated Groups*	*Pass Rate*
Replacement of variable order.	*55*	*55*	*100%*
Reordering of constraint sequences.	*21*	*21*	*100%*
Introduction of multiple new variables in a single linear constraint.	*15*	*15*	*100%*
Introduction of multiple new variables with usage in multiple linear constraints.	*15*	*15*	*100%*
Merging and splitting of constraints.	*6*	*6*	*100%*

CONCLUSION

Conclusion: Contribution

- Introducing the formal paradigm of R1CS constraint sets and designing an algorithm for generating paradigm sets.



- Summarizing the patterns in generating equivalent R1CS constraint sets.
- Designing benchmark for equivalent R1CS constraint sets.

Conclusion: Future Work

- **Merging Rules for Constraints**
  - More Complex Tile Forms
  - Matrices with a Higher Degree of Density
- **More comprehensive test sets**
  - More Comprehensive Categorization
  - Additional and Larger-scale Illustrations.
- **More efficient algorithmic workflow**
  - Improving the Parallelism of the Algorithm.
  - Reducing Algorithm Memory Consumption.

THANK YOU