# Coo: Partial Order Pair Consistency Model

Haixiang Li and Yuxing Chen
{blueseali,axingguchen}@tencent.com

## 1 PRELIMINARY

This section provides the preliminary that will be used and extended in the following section.

*Objects, Operations, Transactions.* We consider storing data **objects** $Obj = \{x, y, ...\}$ in a database. Operations are divided into two groups, i.e., object-oriented operations and state-expressed operations. **Object-oriented operations** are operations on objects by reading or writing. Let $Op_i$ describe the possible invocations set: reading or writing an object by transaction $T_i$. **State-expressed operations** are operations to express states of transactions, consisting of Commit (C) and Abort (A). **Transaction** is a group of operations, interacting objects, with or without a state-expressed operation at the end, representing a committed or an active state. We use subscripts to represent the transaction number. For example, $Op_i[x_n]$ is $x$-oriented operations by transaction $T_i$; $C_i$ and $A_j$ are the committed and abort operations by $T_i$ and $T_j$, respectively.

*Schedules.* An Adya [2] history $H$ comprises a set of transactions $T$ on objects, an order $E$ over operations $Op$ in $T$. The $E$ is persevered the order within a transaction and obeyed the object version order $<_s$. A **schedule** $S$ is a prefix of $H$.

*Example 1.1.* We show an example of a schedule $S_1$ in the following:

$$S_1 = R_1[x_0]\ R_3[x_0]\ W_1[y_1]\ R_3[y_1]\ C_3\ W_2[x_1]\ R_1[y_1]\ A_1. \quad (1)$$

which involves three transactions, where $T_1 = R_1[x]W_1[y]R_1[y]A_1$, $T_2 = W_2[x]$, and $T_3 = R_3[x]R_3[y]C_3$ are aborted, active, and committed transactions respectively. The set of operations is $Op(S_1) = \{R_1[x], R_3[x], W_1[y], R_3[y], W_2[x], R_1[y]\}$. For operations on the same object, we have the version order, e.g., $R_1[x_0] <_s W_2[x_1]$. Note we don't have version order between two reads, e.g., $(R_1[x_0], R_3[x_0])$ or between different objects, e.g, $(R_3[x_0], W_1[y_1])$, meaning reversing these operations may be an *equivalent* schedule.

*Conflict dependency and Conflict graph.* Every history is associated with a conflict graph (also called directed serialization graph) [6, 17], where nodes are committed transactions and edges are the conflicts (read-write, write-write, or write-read) between transactions. The conflict graph is used to test if a schedule is serializable. Intuitively, an acyclic conflict graph indicates a serializable schedule, thus the consistent execution and final state. Figure 1(a) depicts the graphic representation of $S_1$.

## 2 CONSISTENCY MODEL

This section introduces a new consistency model called Coo that can correlate all data anomalies. Specifically, we first proposed *Partial Order Pair (POP) Graph*, which also considers state-expressed operations. We then show any schedule can be represented by a POP graph and our checker can check an anomaly via its POP cycle. Lastly, our generator constructs both centralized and distributed test cases based on POP cycles for the evaluation.

### 2.1 Partial Order Pair Graph

Adya's model introduced some non-cycle anomalies [2, 3] like Dirty Reads and Dirty Write. The reason is that they did not consider state-expressed operations in conflict graph, yet these operations sometimes may be equivalent to object-oriented ones [10]. We strive to map all anomalies via cycles by considering these state-expressed operations. We first formally define POPs as extended conflicts in the following.

*Definition 2.1.* **Partial Order Pair (POP).** Let $T_i$, $T_j$ be transactions in a Schedule $S$ and $T_i \neq T_j$. A Partial Order Pair (POP) is the combination of object-oriented and state-expressed operations from $T_i$ and $T_j$ and satisfies:

- both transactions operate on the same object;
- at least one operation affects the object version (a write or a rollback of a write).

Lemma 2.2. *There exist at most 9 POPs in an arbitrary schedule, i.e.,*$POP = \{WW, WR, RW, WCW, WCR, RCW, RA, WC, WA\}$.

Proof. The proof can be trivially achieved by enumerating all possible combinations of object-oriented and state-expressed operations. Let $T_i, T_j$ be transactions in a Schedule $S$ and $p_i \in T_i$ with $q_j \in T_j$ being object-oriented operations that access the same object, $(p_i, q_j) \in \{W_iW_j, W_iR_j, R_iW_j\}$. The following is a list of all possible combinations.

1. $p_i - q_j$: Both transactions $T_i$ and $T_j$ are still active.

The transaction $T_i$ ends before $T_j$:

2. $p_i - C_i - q_j$: $T_i$ commits before $q_j$;
3. $p_i - A_i - q_j$: $T_i$ aborts before $q_j$;
4. $p_i - q_j - C_i$: $T_i$ commits after $q_j$;
5. $p_i - q_j - A_i$: $T_i$ aborts after $q_j$;

The transaction $T_i$ ends after $T_j$:

6. $p_i - q_j - C_j$: $T_j$ commits after $p_i$;
7. $p_i - q_j - A_j$: $T_j$ aborts after $p_i$.

The operation $p_i$ will not affect the operation $q_j$ in combination 3 due to the timely rollback of $T_i$. So does combination 7. We obtain 15 cases by substituting $\{W_iW_j, R_iW_j, W_iR_j\}$ into $(p_iq_j)$ of the remaining 5 combinations.

Among them, $W_iW_jC_j$ and $W_iW_j$ both have the identical effect of modifying the accessing object by $W_j$, we group them together as POP $WW$. Similarly, we use POP $WR$ to represent $W_iR_j$ and $W_iR_jC_i$ and POP $RW$ to represent $R_iW_j$ and $R_iW_jC_j$. Because read operations are not affected by a commit or abort, we put $R_iW_jA_i$ and $R_iW_jC_i$ into $RW$. Similarly, we put $W_iR_jC_j$ into $WR$. Three cases with committed of $T_i$, i.e., $W_iC_iR_j[x]$, $W_iC_iW_j[x]$, and $R_iC_iW_j[x]$, are specified as types $WCR$, $WCW$, and $RCW$, respectively.

Finally, we have three special combination cases, i.e., $W_iR_jA_i$, $W_iW_jC_i$, and $W_iW_jA_i$, that are more complex as they have two version changing states. As for $W_iR_jA_i$, we have first changing state by $W_iR_j$ then second changing state by $R_jA_i$. $W_iR_j$ belongs to

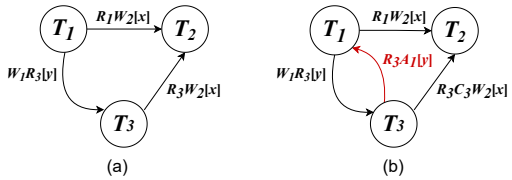


**Figure 1: Comparison of (a) conflict and (b) POP graphs.**

POP $WR$ and $R_jA_i[x]$ belongs to new POP $RA$. Likewise, $W_iW_jC_i$ has $WW$ and $WC$ POPs, and $W_iW_jA_i$ has $WW$ and $WA$ POPs.

In summary, these 15 combination cases are grouped into 9 types POPs, i.e., $WW, WR, RW, WCW, WCR, RCW, RA, WC, WA$. □

Note that RA, WA, and WC are from the combination of a cycle, meaning RA, WA, and WC existed only when the cycle already existed, and this cycle is a 2-transaction cycle on a single object. Let $\mathcal{F} : POP(S) \rightarrow T(S) \times T(S)$ be the map between POPs and the transaction orders, e.g., $\mathcal{F}(W_iC_iR_j[x]) = (T_i, T_j)$. In terms of POPs and their orders, we can define POP graphs.

*Definition 2.3.* **Partial Order Pair Graph (POP graph).** Let $S$ be a schedule. A graph $G(S) = (V, E)$ is called Partial Order Pair Graph (POP graph), if vertices are transactions in $S$ and edges are orders in POPs derived from $S$, i.e (i) $V = T(S)$; (ii) $E = \mathcal{F}(POP(S))$.

Conflict and POP graphs differ in edges and expressiveness. Example 2.4 exemplifies the distinction between them.

*Example 2.4.* Continuing Example 1.1, we obtain objects $Obj$={$x$, $y$}, and operations $Op[x]$={$R_1[x_0]R_3[x_0]C_3W_2[x_1]$} and $Op[y]$= {$W_1[y_1]R_3[y_1]C_3R_1[y_1]A_1$} from $S_1$. Note that we don't put $A_1$ in $Op[x]$ as they don't have a write on object $x$ by $T_1$. We derive POP from these operations, i.e. {$R_1W_2[x], R_3C_3W_2[x]$, $W_1R_3[y]$, $R_3A_1[y]$}. The Conflict graph and the POP graph for $S_1$ are shown in Figure 1. Note that edges from $T_3$ to $T_2$ are different in conflict (RW) and POP (RCW) graph. This time, by a POP graph, the Dirty Read is expressed by a cycle formed by $T_1$ and $T_3$.

Lemma 2.5. *Arbitrary schedules can be represented by POP graphs.*

Proof. Given an arbitrary schedule $S$ with $Op(S)$ being the set of operations by transactions $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$. First, we can derive sets of operations for variables from $S$, $\{OP[x] | x \in Obj(S)\}$. Then we can find all the combination cases in each object operation set $Op[x]$. Finally, we classify them into POPs referred to the proof of Lemma 2.2. Through the above method, we can get the POP set $POP(S)$ corresponding to the schedule $S$. Then, by $\mathcal{F}$, we get the ordering between transactions based on POPs. We can model POP graphs using the transactions set and the dependent orders between transactions. □

## 2.2 Consistency and Consistency Check

With POP cycles, we now are ready to define data anomalies, then define consistency with no data anomaly.

*Definition 2.6.* **Data Anomaly**. The schedule exists a data anomaly exists if the represented POP graph has a cycle.

The definition of data anomalies by POP graphs differs from conflict graph one in three aspects. Firstly, POP graphs model schedules instead of histories (e.g., Full Write in Table 1). Secondly, POP graphs can express all anomalies with state-expressed (e.g., Dirty Read in Definition 2.4). Thirdly, POP graphs can model more distinct anomalies (e.g., Read Skew and Read Skew Committed in Table 1 are different but considered as the same by conflict graph). We now define the consistency of a schedule.

*Definition 2.7.* **Consistency** Schedule $S$ satisfies consistency if the represented POP graph exists no cycle.

*Checker.* By definition 2.7, consistency, no data anomalies, and acyclic POP graphs are equivalent. Likewise, inconsistency, existing data anomalies, and existing POP cycles are equivalent. So a **consistency checker** is to test if a schedule exists a data anomaly, i.e., if the represented graph has a cycle.

In theory, the consistency check is **sound**: if it reports an anomaly in a schedule, then that anomaly should exist in every history of that schedule. The consistency check is **complete**: if it reports an anomaly in a schedule, then a POP cycle exists in the schedule of that anomaly. As a schedule is a prefix of history, the anomaly occurring in the schedule also occurs in the corresponding histories. So the soundness is correct. As we defined that the anomaly schedule exists a POP cycle, the completeness is also correct.

## 2.3 Consistency check in practice.

Table 1 shows all data anomalies types and their classification. The anomaly names with BOLD font are 20+ new types of anomalies that have never been reported (We named them with "committed" when it has a WCW, WCR, or RCW POP). Those reported in Step RAT and Step IAT are a tiny portion of them. Unlike previous tools (e.g., Elle [3]) which randomly issue queries and found anomaly by accident, our generator provides exact sequences of schedules, making the consistency check determined and explainable, meaning it is easy to reproduce and to debug/analyze the result.

Corollary 2.8. *If a schedule satisfies consistency, then the schedule does not have any data anomalies in Table 1.*

The current research mainly focused on centralized databases. There is little research on distributed consistency and it remains ambiguous to do a distributed check. We first define distributed data anomalies.

*Definition 2.9.* **Distributed Data Anomalies** The distributed data anomaly exists if the represented POP graph has a cycle, and it has at least two objects storing at distributed partitions

The **distributed consistency check** is to test if a distributed data anomaly exists. The standard anomalies are not distributed ones and are insufficient for a distributed check as they are single-object. By our classification, we can construct a distributed data anomaly by a DDA or MDA. We particularly designed the test cases to access the different objects from different partitions sometimes from different tables. The design is required by table partitioning and the data is expected to insert/update in different partitions/shards (e.g., by PARTITION BY RANGE in SQL).

# REFERENCES


[1] A. Adya, B. Liskov, and P. O'Neil. 2000. Generalized isolation level definitions. In *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*. 67–78.

[2] Atul Adya and Barbara H. Liskov. 1999. Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions. (1999).

[3] Peter Alvaro and Kyle Kingsbury. 2020. Elle: Inferring Isolation Anomalies from Experimental Observations. *Proc. VLDB Endow.* 14, 3 (2020), 268–280. https://doi.org/10.5555/3430915.3442427

[4] Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil. 1995. A Critique of ANSI SQL Isolation Levels. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data* (San Jose, California, USA) *(SIGMOD '95)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/223784.223785

[5] Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. 1995. A Critique of ANSI SQL Isolation Levels. In *SIGMOD Conference*. ACM Press, 1–10.

[6] Philip A. Bernstein and Nathan Goodman. 1983. Multiversion Concurrency Control - Theory and Algorithms. *ACM Trans. Database Syst.* 8, 4 (1983), 465–483.

[7] Carsten Binnig, Stefan Hildenbrand, Franz Farber, Donald Kossmann, Juchang Lee, and Norman May. 2014. Distributed snapshot isolation: global transactions pay globally, local transactions pay locally. 23, 6 (2014), 987–1011.

[8] Sebastian Burckhardt, Daan Leijen, Jonathan Protzenko, and Manuel Fähndrich. 2015. Global Sequence Protocol: A Robust Abstraction for Replicated Shared State. In *29th European Conference on Object-Oriented Programming (ECOOP 2015) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 37)*, John Tang Boyland (Ed.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 568–590. https://doi.org/10.4230/LIPIcs.ECOOP.2015.568

[9] Andrea Cerone, Alexey Gotsman, and Hongseok Yang. 2017. Algebraic Laws for Weak Consistency. (2017), 26:1–26:18.

[10] Natacha Crooks, Youer Pu, Lorenzo Alvisi, and Allen Clement. 2017. Seeing is Believing: A Client-Centric Specification of Database Isolation. In *PODC*. ACM, 73–82.

[11] Alan Fekete, Elizabeth O'Neil, and Patrick O'Neil. 2004. A Read-Only Transaction Anomaly under Snapshot Isolation. *SIGMOD Rec.* 33, 3 (Sept. 2004), 12–14. https://doi.org/10.1145/1031570.1031573

[12] American National Standard for Information Systems – Database Language. Nov 1992. ANSI X3.135-1992. SQL.

[13] Zhenghua Lyu, Huan Hubert Zhang, Gang Xiong, Gang Guo, Haozhou Wang, Jinbao Chen, Asim Praveen, Yu Yang, Xiaoming Gao, Alexandra Wang, Wen Lin, Ashwin Agrawal, Junfeng Yang, Hao Wu, Xiaoliang Li, Feng Guo, Jiang Wu, Jesse Zhang, and Venkatesh Raghavan. 2021. Greenplum: A Hybrid Database for Transactional and Analytical Workloads. In *SIGMOD Conference*. ACM, 2530–2542.

[14] Ralf Schenkel, Gerhard Weikum, N Weissenberg, and Xuequn Wu. 2000. Federated transaction management with snapshot isolation. *Lecture Notes in Computer Science* (2000), 1–25.

[15] wikipedia. 2022. Read_Only_Transactions. https://wiki.postgresql.org/wiki/SSI#Read_Only_Transactions

[16] Chao Xie, Chunzhi Su, Cody Littley, Lorenzo Alvisi, Manos Kapritsos, and Yang Wang. 2015. High-performance ACID via modular concurrency control. In *Proceedings of the 25th Symposium on Operating Systems Principles*. 279–294.

[17] Maysam Yabandeh and Daniel Gómez Ferro. 2012. A critique of snapshot isolation. In *EuroSys*. ACM, 155–168.


**Table 1: Data anomaly formal expression, classification, and their (Partial Order Pair) POP combinations in POP cycles.**

| Types of Anomalies | | No | Anomalies | Formal expressions | POP Combinations |
|---|---|---|---|---|---|
| RAT | SDA | 1 | Dirty Read [1, 12, 16] | $W_i[x_m]\dots R_j[x_m]\dots A_i$ | $W_iR_j[x]-R_jA_i[x]$ |
| | SDA | 2 | Non-repeatable Read [12] | $R_i[x_m]\dots W_j[x_{m+1}]\dots R_i[x_{m+1}]$ | $R_iW_j[x]-W_jR_i[x]$ |
| | SDA | 3 | Intermediate Read [1, 16] | $W_i[x_m]\dots R_j[x_m]\dots W_i[x_{m+1}]$ | $W_iR_j[x]-R_jW_i[x]$ |
| | SDA | 4 | **Intermediate Read Committed** | $W_i[x_m]\dots R_j[x_m]\dots C_j\dots W_i[x_{m+1}]$ | $W_iR_j[x]-R_jC_jW_i[x]$ |
| | SDA | 5 | **Lost Self Update** | $W_i[x_m]\dots W_j[x_{m+1}]\dots R_i[x_{m+1}]$ | $W_iW_j[x]-W_jR_i[x]$ |
| | DDA | 6 | **Write-read Skew** | $W_i[x_m]\dots R_j[x_m]\dots W_j[y_n]\dots R_i[y_n]$ | $W_iR_j[x]-W_jR_i[y]$ |
| | DDA | 7 | **Write-read Skew Committed** | $W_i[x_m]\dots R_j[x_m]\dots W_j[y_n]\dots C_j\dots R_i[y_n]$ | $W_iR_j[x]-W_jC_jR_i[y]$ |
| | DDA | 8 | **Double-write Skew 1** | $W_i[x_m]\dots R_j[x_m]\dots W_j[y_n]\dots W_i[y_{n+1}]$ | $W_iR_j[x]-W_jW_i[y]$ |
| | DDA | 9 | **Double-write Skew 1 Committed** | $W_i[x_m]\dots R_j[x_m]\dots W_j[y_n]\dots C_j\dots W_i[y_{n+1}]$ | $W_iR_j[x]-W_jC_jW_i[y]$ |
| | DDA | 10 | **Double-write Skew 2** | $W_i[x_m]\dots W_j[x_{m+1}]\dots W_j[y_n]\dots R_i[y_n]$ | $W_iW_j[x]-W_jR_i[y]$ |
| | DDA | 11 | Read Skew [4] | $R_i[x_m]\dots W_j[x_{m+1}]\dots W_j[y_n]\dots R_i[y_n]$ | $R_iW_j[x]-W_jR_i[y]$ |
| | DDA | 12 | **Read Skew 2** | $W_i[x_m]\dots R_j[x_m]\dots R_j[y_n]\dots W_i[y_{n+1}]$ | $W_iR_j[x]-R_jW_i[y]$ |
| | DDA | 13 | **Read Skew 2 Committed** | $W_i[x_m]\dots R_j[x_m]\dots R_j[y_n]\dots C_j\dots W_i[y_{n+1}]$ | $W_iR_j[x]-R_jC_jW_i[y]$ |
| | MDA | 14 | **Step RAT** [8, 9] | $\dots W_i[x_m]\dots R_j[x_m]\dots$, and $N_{obj}\geq 2, N_T\geq 3$ | $\dots W_iR_j[x]\dots$ |
| WAT | SDA | 15 | Dirty Write [12] | $W_i[x_m]\dots W_j[x_{m+1}]\dots A_i/C_i$ | $W_iW_j[x]-W_jA_i/C_i[x]$ |
| | SDA | 16 | **Full Write** | $W_i[x_m]\dots W_j[x_{m+1}]\dots W_i[x_{m+2}]$ | $W_iW_j[x]-W_jW_i[x]$ |
| | SDA | 17 | **Full Write Committed** | $W_i[x_m]\dots W_j[x_{m+1}]\dots C_j\dots W_i[x_{m+2}]$ | $W_iW_j[x]-W_jC_jW_i[x]$ |
| | SDA | 18 | Lost Update [4] | $R_i[x_m]\dots W_j[x_{m+1}]\dots W_i[x_{m+2}]$ | $R_iW_j[x]-W_jW_i[x]$ |
| | SDA | 19 | **Lost Self Update Committed** | $W_i[x_m]\dots W_j[x_{m+1}]\dots C_j\dots R_i[x_{m+1}]$ | $W_iW_j[x]-W_jC_jR_i[x]$ |
| | DDA | 20 | **Double-write Skew 2 Committed** | $W_i[x_m]\dots W_j[x_{m+1}]\dots W_j[y_n]\dots C_j\dots R_i[y_n]$ | $W_iW_j[x]-W_jC_jR_i[y]$ |
| | DDA | 21 | Full-write Skew [13] | $W_i[x_m]\dots W_j[x_{m+1}]\dots W_j[y_n]\dots W_i[y_{n+1}]$ | $W_iW_j[x]-W_jW_i[y]$ |
| | DDA | 22 | **Full-write Skew Committed** | $W_i[x_m]\dots W_j[x_{m+1}]\dots W_j[y_n]\dots C_j\dots W_i[y_{n+1}]$ | $W_iW_j[x]-W_jC_jW_i[y]$ |
| | DDA | 23 | **Read-write Skew 1** | $R_i[x_m]\dots W_j[x_{m+1}]\dots W_j[y_n]\dots W_i[y_{n+1}]$ | $R_iW_j[x]-W_jW_i[y]$ |
| | DDA | 24 | **Read-write Skew 2** | $W_i[x_m]\dots W_j[x_{m+1}]\dots R_j[y_n]\dots W_i[y_{n+1}]$ | $W_iW_j[x]-R_jW_i[y]$ |
| | DDA | 25 | **Read-write Skew 2 Committed** | $W_i[x_m]\dots W_j[x_{m+1}]\dots R_j[y_n]\dots C_j\dots W_i[y_{n+1}]$ | $W_iW_j[x]-R_jC_jW_i[y]$ |
| | MDA | 26 | **Step WAT** | $\dots W_i[x_m]\dots W_j[x_{m+1}]\dots$, and $N_{obj}\geq 2, N_T\geq 3$, and not include $(\dots W_{i1}[y_n]\dots R_{j1}[y_n]\dots)$ | $\dots W_iW_j[x]\dots$ |
| IAT | SDA | 27 | Non-repeatable Read Committed [12] | $R_i[x_m]\dots W_j[x_{m+1}]\dots C_j\dots R_i[x_{m+1}]$ | $R_iW_j[x]-W_jC_jR_i[x]$ |
| | SDA | 28 | **Lost Update Committed** | $R_i[x_m]\dots W_j[x_{m+1}]\dots C_j\dots W_i[x_{m+2}]$ | $R_iW_j[x]-W_jC_jW_i[x]$ |
| | DDA | 29 | Read Skew Committed [4] | $R_i[x_m]\dots W_j[x_{m+1}]\dots W_j[y_n]\dots C_j\dots R_i[y_n]$ | $R_iW_j[x]-W_jC_jR_i[y]$ |
| | DDA | 30 | **Read-write Skew 1 Committed** | $R_i[x_m]\dots W_j[x_{m+1}]\dots W_j[y_n]\dots C_j\dots W_i[y_{n+1}]$ | $R_iW_j[x]-W_jC_jW_i[y]$ |
| | DDA | 31 | Write Skew [5] | $R_i[x_m]\dots W_j[x_{m+1}]\dots R_j[y_n]\dots W_i[y_{n+1}]$ | $R_iW_j[x]-R_jW_i[y]$ |
| | DDA | 32 | **Write Skew Committed** | $R_i[x_m]\dots W_j[x_{m+1}]\dots R_j[y_n]\dots C_j\dots W_i[y_{n+1}]$ | $R_iW_j[x]-R_jC_jW_i[y]$ |
| | MDA | 33 | **Step IAT** [7, 9, 11, 14, 15] | Not include $(\dots W_{i1}[x_m]\dots R_{j1}[x_m]\dots$ and $\dots W_{i2}[y_n]\dots W_{j2}[y_{n+1}]\dots)$, $N_{obj}\geq 2, N_T\geq 3$ | $\dots R_iW_j[x]\dots$ |