PROTOKOLL

im Studiengang Mechatronik/Robotik (Master)
Lehrveranstaltung Moderne Programmierkonzepte (VMPK)

Lösung eines Megaminx in Prolog

Ausgeführt von:
Dominik Diek (mr20m047)
Johannes Honigschnabl (mr20m035)
Yannick Konstandin (mr20m012)
Martin Kritzl (mr20m037)
Maximilian Siegl (mr20m020)
Philipp Wolf (mr20m027)

Begutachter: FH-Prof. Dipl. Ing. Dr. Lars Mehnen

Wien, 31.01.2021

FH University of Applied Sciences
TECHNIKUM WIEN

Inhaltsverzeichnis

1 Einleitung

Der *Megaminx* ist eine 12-seitige Variante des altbewährten 3x3-Rubiks-Cubes. Dieser soll mit Hilfe eines Programmes gelöst werden. Die Implementation des Lösungsansatzes soll in Prolog erfolgen. Für die Lösung des Megaminx soll eine Heuristik verwendet werden, welche es erlaubt den Würfel mit möglichst wenigen Zügen in die richtige Konstellation zu drehen. Dabei soll auch zwischen zwei Versionen unterschieden werden. Eine Ausführung mit vorgegebenen Zwischenzielen („menschliche Lösung“) soll mit einem Algorithmus (z.B.: RBFS) verglichen werden.

2 Prinzip

Da der Würfel ca. 10^{68} mögliche Stellungen einnehmen kann, ist es nicht möglich alle durchzuprobieren, da sonst die Rechenzeit zu groß wäre. Daher ist es notwendig eine Art von Suchalgorithmus zu implementieren, um die Lösung schneller zu finden. Hierbei wird ein Best-first search (Bestensuche) angewendet, welcher einen Graphen durchsucht und hierbei immer den Weg mit der besten Aussicht wählt. Diese Metrik, die verschiedenen Möglichkeiten zu gewichten, nennt man Heuristik. Je nach Qualität der Heuristik richtet sich dann auch die Geschwindigkeit des Suchalgorithmus. Im Folgenden werden nun mehrere Ansätze für die Implementation einer solchen Heuristik mit der Suche beschrieben.

2.1 Pattern-Databases

Die Heuristik, die oft bei der Lösung eines 3x3x3 Würfels angewandt wird, ist die Verwendung von Pattern-Databases. Pattern-Databases können als eine Sammlung von Lösungen für Teilziele betrachtet werden, die erreicht werden müssen, um das Problem zu lösen. Die Grundidee beim Lösen eine Rubiks Cube besteht darin, dass für einen Stein eine bestimmte Anzahl von Zügen möglich sind, um an die gewollte Endposition zu gelangen. All diese Teillösungen sind in den Pattern-Databases gespeichert und das für jeden einzelnen Stein. Bei einem 3x3x3 Rubiks resultiert dann eine Datenbank mit Unmengen an Speicherplatz, welche zuerst generiert und nachfolgend noch ausgewertet wird, um auf eine Lösung zu gelangen.

Bei einem Megaminx mit zwölf Seiten wird die Datenbank folglich auch noch deutlich größer und würde bei Weitem die Speicherkapazität eines normalen Arbeits-PCs übersteigen. Außerdem ist die Umsetzung in Prolog auch nicht bekannt.

2.2 Geometrische Heuristik

Ein weiterer Lösungsansatz ist die Verwendung des geometrischen Abstands eines einzelnen Steins zu seiner Endposition. Hierzu ist der kürzeste Abstand, eine Art Luftlinie, zu verwenden oder die

Manhatten-Metrik anzunehmen, welche die Summe der absoluten Differenzen ihrer Einzelkoordinaten definiert. Mithilfe dieses, im Optimalfall euklidischen, Abstands in mm kann dann eine Heuristik implementiert werden.

Den Abstand ausrechnen zu lassen und in Verbindung mit Prolog ein ausführbares Programm zu schreiben wurde im Rahmen dieses Projekts als nicht möglich eingeschätzt.

2.3 Korrekte Flächen

Die einfachste Denkweise einer Heuristik bei einem derartigen Beispiel ist die des Matchings. Bei jeder Rotation wird die aktuelle Stellung mit der Lösung verglichen. Je mehr Flächen bereits die korrekte Position haben, desto besser. Ein Algorithmus um diese beiden Stellungen (Arrays) zu vergleichen ist in Anhang A angegeben.

Die Problematik hierbei ist jedoch, dass falsche Positionierungen trotzdem zu einer guten Kostenfunktion führen. Wenn zum Beispiel ein weißer Kantenstein mit der weißen Fläche neben dem weißen Mittelstück liegt, jedoch alle anderen Farben nicht übereinstimmen, so ist trotzdem eine Übereinstimmung vorhanden, obwohl der Stein vielleicht auf der falschen Seite liegt.

Außerdem kann keine dynamische Ermittlung der Kosten stattfinden. Entweder eine Fläche ist richtig oder nicht, mehr Information erhält man nicht. Lediglich über die Summe der korrekten Flächen entsteht eine gewisse Dynamik. Daher ist diese Methode keine gute Metrik.

2.4 Heuristik-Tabelle

Eine weitere Möglichkeit einer Heuristik ist eine simple Tabelle mit den Abständen einer Fläche zum jeweiligen Mittelstück der gleichen Farbe. Dies ist auch die Methode, welche wir zur Implementation in Prolog wählen würden. Sie ähnelt sehr der Methode mit Pattern-Databases, mit dem Unterschied, dass hier keine berechneten Tabellen verwendet werden, sondern nur eine Tabelle, welche händisch angefertigt wurde.

Eine solche Tabelle ist in Code 1 angegeben und beinhaltet ein Array, in dem der Abstand einer Fläche zur Up-Seite angegeben ist. Der Abstand ist hierbei die minimale Anzahl an Sprüngen von einer Fläche zur nächsten, um vom Mittelstein der oberen Seite zum gegebenen Stein zu gelangen. Da die Mittelsteine ihre Position relativ zueinander nicht verändern können, brauchen diese keine Heuristik und haben alle den Wert 0 (rechte Spalte in der Tabelle).

```
cube(
    %1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10, 11
     1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 0 , % Up / weiss
     2 , 2 , 3 , 4 , 5 , 6 , 5 , 4 , 3 , 2 , 0 , % Front / dunkel gruen
     3 , 4 , 5 , 6 , 5 , 4 , 3 , 2 , 2 , 2 , 0 , % Left / violett
     3 , 2 , 2 , 2 , 3 , 4 , 5 , 6 , 5 , 4 , 0 , % Right / rot
     6 , 7 , 8 , 9 , 10, 9 , 8 , 7 , 6 , 5 , 0 , % Front left down / hell blau
     6 , 5 , 6 , 7 , 8 , 9 , 10, 9 , 8 , 7 , 0 , % Front right down / braun
     11, 10, 11, 10, 11, 10, 11, 10, 11, 10, 0 , % Down / grau
     10, 9 , 8 , 7 , 6 , 5 , 6 , 7 , 8 , 9 , 0 , % Back / hell gruen
     8 , 7 , 6 , 5 , 6 , 7 , 8 , 9 , 10, 9 , 0 , % Back left down / orange
     8 , 9 , 10, 9 , 8 , 7 , 6 , 5 , 6 , 7 , 0 , % Back right down / rosa
     5 , 4 , 3 , 2 , 2 , 2 , 3 , 4 , 5 , 6 , 0 , % Back left up / gelb
     5 , 6 , 5 , 4 , 3 , 2 , 2 , 2 , 3 , 4 , 0 ) % Back right up / dunkel blau
    ).
```

Code 1: Heuristik-Tabelle

Die Up-Seite ist in diesem Fall jene Seite, die sich oben am Würfel befindet. Um nun die Heuristik einer einzelnen Farbe zu ermitteln, muss die aktuelle Position mit der Tabelle verglichen und die korrekten Werte herausgelesen werden.

Um dies zu erklären, wird hier als Beispiel die Heuristik der Farbe Weiß ermittelt. Der erste Schritt ist es, den Würfel so zu drehen, dass sich das weiße Mittelstück auf der Oberseite befindet. Nun werden die Positionen aller weißen Flächen ermittelt. Die Werte aus der Heuristik-Tabelle an genau diesen Positionen werden dann ermittelt und addiert. Somit erhält man eine gesamte Metrik für die weißen Flächen. Diesen Vorgang wiederholt man für alle anderen Farben auch, indem man die jeweilige Seite nach oben dreht und addiert alle Ergebnisse. Dieser Wert gibt nun die Heuristik für die aktuelle Stellung an. Durch diese Methode ergibt sich für jede Fläche ein Wert von 1 bis 11 und somit auch mehr Dynamik als das Beispiel mit den korrekten Flächen.

Diese Heuristik kann dann für den RBFS-Algorithmus verwendet werden. Hierbei probiert der Algorithmus in jedem Schritt alle möglichen Drehungen des Megaminx aus und ermittelt dann für jede Verdrehung mit der Heuristik, ob der Megaminx sich der Lösung nähert. Der Weg mit dem geringsten Wert der Heuristik, und somit der besten Aussicht, wird gewählt und weiterverfolgt. Am Ende ergibt sich eine Folge von Drehungen, welche lediglich rückverfolgt werden, und so zur Lösung des Würfels führt.

3 A*

Die oben erwähnten Heuristiken können nun auch mit einem A*-Suchalgorithmus kombiniert werden. Somit könnte eine optimale Lösung gefunden werden, wenn die Heuristik die Kosten nicht unterschätzt. Die Schwierigkeit dies in Prolog umzusetzen ist jedoch, eine sinnvolle Kostenfunktion zu implementieren. Eine der Möglichkeiten ist es, die getätigten Drehungen für jeden Stein zu tracken. Dann wäre man wieder an der Problematik, welche bereits im Kapitel 2.1 beschrieben wurden, dass solche Berechnungen nicht während der Laufzeit durchgeführt werden können, sondern in Datenbanken abgespeichert werden.

4 Vergleich mit „menschlicher“ Lösung

Verglichen mit der menschlichen Lösung hat die maschinelle Lösung einige Vorteile. Die Lösung kann schneller gefunden werden, da eine Metrik vorliegt, welche den Suchbereich einschränkt und nur Lösungen verfolgt, welche den Megaminx einer Lösung näherbringen. Im Gegensatz dazu verfolgt die menschliche Lösung alle Möglichkeiten und probiert so lange, bis eine Lösung gefunden wird. Daher sind dort auch Zwischenziele notwendig, um die Suchkette zu verringern. Dies erhöht wieder den Aufwand für den Programmierer, da sehr viel manuelle Definitionen der Zwischenziele und mögliche Rotationen jedes Zieles durchzuführen sind. Dies alles schlägt sich auch in der benötigten Rechenkapazität nieder, da eben schneller ein Ziel gefunden wird und somit weniger Rechenzeit notwendig ist.

Außerdem ist die Lösung des Suchalgorithmus besser optimiert als die menschliche Lösung. Es wird ein direkter Weg zum Ziel gesucht, während bei der menschlichen Lösung eine Reihe von Bewegungen dazu führen können, dass bereits gelöste Seiten wieder verdreht werden können. Dies lässt sich dadurch beobachten, dass bei der menschlichen Lösung anfängliche Verdrehungen mit nur einer geringen Zahl an Rotationen (z.B. 3) zu einer langen Rechenzeit und Lösung führen (z.B. 20 Rotationen oder mehr).

Durch diese Faktoren eignet sich ein Suchalgorithmus besser zur Lösung eines Megaminx.

5 Resümee

Bei der Lösung eines Megaminx ist man vor allem bestrebt eine Lösung zu finden, die wenig Zeit benötigt. Gerade im solchen Situationen helfen Suchalgorithmen dabei, die Rechenzeit zu minimieren. Zwar bekommt man mit der *händischen* Umsetzung auch immer eine Lösung, jedoch kann die dafür benötigte Zeit stark schwanken. Zusätzlich ist der Programmieraufwand für eine solche Lösung enorm, da sämtliche Zwischenziele definiert werden müssen.

Ein effizienterer Lösungsansatz wird mit Hilfe einer Heuristik-Tabelle erzielt. Der Vorteil hierbei liegt darin, dass durch die Kenntnis der Abstände die Lösungswege vorher verglichen werden können und somit der Weg ausgewählt werden kann, welcher die wenigsten Drehungen benötigt. Dadurch kann die Rechenzeit enorm gesenkt werden. Es ist jedoch nicht ganz trivial einen solchen Algorithmus auf ein bestimmtes Problem anzuwenden. Dieser Weg kann sogar noch effizienter gemacht werden, wenn man die Heuristik-Tabelle mit einem A*-Algorithmus kombiniert. Das bringt jedoch auch Probleme mit sich, da die Berechnungen beim Tracken nicht während der Laufzeit durchgeführt werden können. Man benötigt also somit wieder mehr Speicher bei der Berechnung. Je nach Hardware ist es demnach durchaus sinnvoll einen A*-Algorithmus zusätzlich zu verwenden, um eine effiziente und zielbringende Lösung zu erhalten.

6 Literaturverzeichnis

Arfaee S., Z. S. (2011). *Learning heuristics functions for large state spaces.*

B., B. (2020). *Implementing an Optimal Rubik's Cube Solver using Korfs Algorithm. [Online]. Verfügbar unter: <https://medium.com/@benjamin.botto/implementing-an-optimal-rubiks-cube-solver-using-korf-s-algorithm-bf750b332cf9> [Zugang am 31.01.2021].*

CPP.EDU. (2010). *The A* algorithm in Prolog. [Online]. Verrfügbar unter: <https://www.cpp.edu/~jrfisher/www/prolog_tutorial/5_1.html> [Zugang am 31.01.2021].*

Demaine, E. D. (2011). *Algorithms for Solving Rubik's Cubes.*

Korf, R. (2006). *Finding Optimal Solutions to Rubik's Cube Using Pattern Databases.*

Norvig, R. &. (2018). *Artificial Intelligence Week 4: Informed Search (Chapter 3).*

Tek-TIps. (2010). *Counting the same elements in the list. [Online]. Verfügbar unter: <https://www.tek-tips.com/viewthread.cfm?qid=1604418> [Zugang am 31.01.2021].*

Weed, J. (2016). *Sub-Optimal Multi-Phase Path Planning: A Method for Solving Rubik's Revenge.*

Wikipedia. (2021). *A* search algorithm [Online]. Verfügbar unter: <https://en.wikipedia.org/wiki/A*_search_algorithm> [Zugang am 31.01.2021].*

Wikipedia. (2021). *Optimal solutions for Rubik's Cube [Online]. <Verfügbar unter: https://en.wikipedia.org/wiki/Optimal_solutions_for_Rubik%27s_Cube> [Zugang am 31.01.2021].*

Zeng, W. C. (2007). *Shortest paths and A*. Finding shortest paths on real road networks.*

Anhang A: Prolog-Code zum Bestimmen gleicher Elemente in einem Array

```
count_common_elements([],[],0).
count_common_elements([X|T1],[X|T2],Y):-
     count_common_elements(T1,T2,Z), Y is 1+Z.
count_common_elements([X1|T1],[X2|T2],Z):-
     X1\=X2,count_common_elements(T1,T2,Z).

heuristic(
    megaminx(
    W1_0  ,W1_1  ,W1_2  ,W1_3  ,W1_4  ,W1_5  ,W1_6  ,W1_7  ,W1_8  ,W1_9  ,W1_10 ,
    G1_0  ,G1_1  ,G1_2  ,G1_3  ,G1_4  ,G1_5  ,G1_6  ,G1_7  ,G1_8  ,G1_9  ,G1_10 ,
    R1_0  ,R1_1  ,R1_2  ,R1_3  ,R1_4  ,R1_5  ,R1_6  ,R1_7  ,R1_8  ,R1_9  ,R1_10 ,
    B1_0  ,B1_1  ,B1_2  ,B1_3  ,B1_4  ,B1_5  ,B1_6  ,B1_7  ,B1_8  ,B1_9  ,B1_10 ,
    Y1_0  ,Y1_1  ,Y1_2  ,Y1_3  ,Y1_4  ,Y1_5  ,Y1_6  ,Y1_7  ,Y1_8  ,Y1_9  ,Y1_10 ,
    Pr1_0 ,Pr1_1 ,Pr1_2 ,Pr1_3 ,Pr1_4 ,Pr1_5 ,Pr1_6 ,Pr1_7 ,Pr1_8 ,Pr1_9 ,Pr1_10,
    H1_0  ,H1_1  ,H1_2  ,H1_3  ,H1_4  ,H1_5  ,H1_6  ,H1_7  ,H1_8  ,H1_9  ,H1_10 ,
    Pi1_0 ,Pi1_1 ,Pi1_2 ,Pi1_3 ,Pi1_4 ,Pi1_5 ,Pi1_6 ,Pi1_7 ,Pi1_8 ,Pi1_9 ,Pi1_10,
    L1_0  ,L1_1  ,L1_2  ,L1_3  ,L1_4  ,L1_5  ,L1_6  ,L1_7  ,L1_8  ,L1_9  ,L1_10 ,
    O1_0  ,O1_1  ,O1_2  ,O1_3  ,O1_4  ,O1_5  ,O1_6  ,O1_7  ,O1_8  ,O1_9  ,O1_10 ,
    T1_0  ,T1_1  ,T1_2  ,T1_3  ,T1_4  ,T1_5  ,T1_6  ,T1_7  ,T1_8  ,T1_9  ,T1_10 ,
    Gr1_0 ,Gr1_1 ,Gr1_12 ,Gr1_3 ,Gr1_4 ,Gr1_5 ,Gr1_6 ,Gr1_7 ,Gr1_8 ,Gr1_9 ,Gr1_10),

    megaminx(
    W2_0  ,W2_1  ,W2_2  ,W2_3  ,W2_4  ,W2_5  ,W2_6  ,W2_7  ,W2_8  ,W2_9  ,W2_10 ,
    G2_0  ,G2_1  ,G2_2  ,G2_3  ,G2_4  ,G2_5  ,G2_6  ,G2_7  ,G2_8  ,G2_9  ,G2_10 ,
    R2_0  ,R2_1  ,R2_2  ,R2_3  ,R2_4  ,R2_5  ,R2_6  ,R2_7  ,R2_8  ,R2_9  ,R2_10 ,
    B2_0  ,B2_1  ,B2_2  ,B2_3  ,B2_4  ,B2_5  ,B2_6  ,B2_7  ,B2_8  ,B2_9  ,B2_10 ,
    Y2_0  ,Y2_1  ,Y2_2  ,Y2_3  ,Y2_4  ,Y2_5  ,Y2_6  ,Y2_7  ,Y2_8  ,Y2_9  ,Y2_10 ,
    Pr2_0 ,Pr2_1 ,Pr2_2 ,Pr2_3 ,Pr2_4 ,Pr2_5 ,Pr2_6 ,Pr2_7 ,Pr2_8 ,Pr2_9 ,Pr2_10,
    H2_0  ,H2_1  ,H2_2  ,H2_3  ,H2_4  ,H2_5  ,H2_6  ,H2_7  ,H2_8  ,H2_9  ,H2_10 ,
    Pi2_0 ,Pi2_1 ,Pi2_2 ,Pi2_3 ,Pi2_4 ,Pi2_5 ,Pi2_6 ,Pi2_7 ,Pi2_8 ,Pi2_9 ,Pi2_10,
    L2_0  ,L2_1  ,L2_2  ,L2_3  ,L2_4  ,L2_5  ,L2_6  ,L2_7  ,L2_8  ,L2_9  ,L2_10 ,
    O2_0  ,O2_1  ,O2_2  ,O2_3  ,O2_4  ,O2_5  ,O2_6  ,O2_7  ,O2_8  ,O2_9  ,O2_10 ,
    T2_0  ,T2_1  ,T2_2  ,T2_3  ,T2_4  ,T2_5  ,T2_6  ,T2_7  ,T2_8  ,T2_9  ,T2_10 ,
    Gr2_0 ,Gr2_1 ,Gr2_2 ,Gr2_3 ,Gr2_4 ,Gr2_5 ,Gr2_6 ,Gr2_7 ,Gr2_8 ,Gr2_9 ,Gr2_10),
    C):-
        count_common_elements(
            [
               W1_0  ,W1_1  ,W1_2  ,W1_3  ,W1_4  ,W1_5  ,W1_6  ,W1_7  ,W1_8  ,W1_9  ,W1_10 ,
               G1_0  ,G1_1  ,G1_2  ,G1_3  ,G1_4  ,G1_5  ,G1_6  ,G1_7  ,G1_8  ,G1_9  ,G1_10 ,
               R1_0  ,R1_1  ,R1_2  ,R1_3  ,R1_4  ,R1_5  ,R1_6  ,R1_7  ,R1_8  ,R1_9  ,R1_10 ,
               B1_0  ,B1_1  ,B1_2  ,B1_3  ,B1_4  ,B1_5  ,B1_6  ,B1_7  ,B1_8  ,B1_9  ,B1_10 ,
               Y1_0  ,Y1_1  ,Y1_2  ,Y1_3  ,Y1_4  ,Y1_5  ,Y1_6  ,Y1_7  ,Y1_8  ,Y1_9  ,Y1_10 ,
               Pr1_0 ,Pr1_1 ,Pr1_2 ,Pr1_3 ,Pr1_4 ,Pr1_5 ,Pr1_6 ,Pr1_7 ,Pr1_8 ,Pr1_9 ,Pr1_10,
               H1_0  ,H1_1  ,H1_2  ,H1_3  ,H1_4  ,H1_5  ,H1_6  ,H1_7  ,H1_8  ,H1_9  ,H1_10 ,
               Pi1_0 ,Pi1_1 ,Pi1_2 ,Pi1_3 ,Pi1_4 ,Pi1_5 ,Pi1_6 ,Pi1_7 ,Pi1_8 ,Pi1_9 ,Pi1_10,
               L1_0  ,L1_1  ,L1_2  ,L1_3  ,L1_4  ,L1_5  ,L1_6  ,L1_7  ,L1_8  ,L1_9  ,L1_10 ,
               O1_0  ,O1_1  ,O1_2  ,O1_3  ,O1_4  ,O1_5  ,O1_6  ,O1_7  ,O1_8  ,O1_9  ,O1_10 ,
               T1_0  ,T1_1  ,T1_2  ,T1_3  ,T1_4  ,T1_5  ,T1_6  ,T1_7  ,T1_8  ,T1_9  ,T1_10 ,
               Gr1_0 ,Gr1_1 ,Gr1_12 ,Gr1_3 ,Gr1_4 ,Gr1_5 ,Gr1_6 ,Gr1_7 ,Gr1_8 ,Gr1_9 ,Gr1_10
            ],

            [
               W2_0  ,W2_1  ,W2_2  ,W2_3  ,W2_4  ,W2_5  ,W2_6  ,W2_7  ,W2_8  ,W2_9  ,W2_10 ,
               G2_0  ,G2_1  ,G2_2  ,G2_3  ,G2_4  ,G2_5  ,G2_6  ,G2_7  ,G2_8  ,G2_9  ,G2_10 ,
               R2_0  ,R2_1  ,R2_2  ,R2_3  ,R2_4  ,R2_5  ,R2_6  ,R2_7  ,R2_8  ,R2_9  ,R2_10 ,
               B2_0  ,B2_1  ,B2_2  ,B2_3  ,B2_4  ,B2_5  ,B2_6  ,B2_7  ,B2_8  ,B2_9  ,B2_10 ,
               Y2_0  ,Y2_1  ,Y2_2  ,Y2_3  ,Y2_4  ,Y2_5  ,Y2_6  ,Y2_7  ,Y2_8  ,Y2_9  ,Y2_10 ,
               Pr2_0 ,Pr2_1 ,Pr2_2 ,Pr2_3 ,Pr2_4 ,Pr2_5 ,Pr2_6 ,Pr2_7 ,Pr2_8 ,Pr2_9 ,Pr2_10,
               H2_0  ,H2_1  ,H2_2  ,H2_3  ,H2_4  ,H2_5  ,H2_6  ,H2_7  ,H2_8  ,H2_9  ,H2_10 ,
               Pi2_0 ,Pi2_1 ,Pi2_2 ,Pi2_3 ,Pi2_4 ,Pi2_5 ,Pi2_6 ,Pi2_7 ,Pi2_8 ,Pi2_9 ,Pi2_10,
               L2_0  ,L2_1  ,L2_2  ,L2_3  ,L2_4  ,L2_5  ,L2_6  ,L2_7  ,L2_8  ,L2_9  ,L2_10 ,
               O2_0  ,O2_1  ,O2_2  ,O2_3  ,O2_4  ,O2_5  ,O2_6  ,O2_7  ,O2_8  ,O2_9  ,O2_10 ,
               T2_0  ,T2_1  ,T2_2  ,T2_3  ,T2_4  ,T2_5  ,T2_6  ,T2_7  ,T2_8  ,T2_9  ,T2_10 ,
               Gr2_0 ,Gr2_1 ,Gr2_2 ,Gr2_3 ,Gr2_4 ,Gr2_5 ,Gr2_6 ,Gr2_7 ,Gr2_8 ,Gr2_9 ,Gr2_10],
            C).
```