L2/22-228

Feedback on IRGN2572 “Proposal to encode 5 new ideograph description characters”

Author: FAN Ming
Date: 2022.09.02 (Revised)
Status: Individual Contribution

Many new IDCs were proposed in IRGN2572 for encoding, and some of them, especially unary operators (⿾,⿿) and the subtraction operator (㇯) were concerned by many that they might added ambiguity of IDS representation and are hostile to computer algorithms. However, I argue that most problems regarding ambiguity can at least be resolved theoretically by algorithms, and this article is the technical details of these solutions.

1. Rotation and Reflection (⿾ and ⿿)

In IRGN2572SATFeedback, there are mainly two issues raised by SAT regarding ambiguity of the rotation and the reflection operator, that is: 1) Issue of using sequences as argument of these operator, 2) Consecutive usage of these operators. Here I will show that all these two issues can be solved at least theoretically.

1) Consecutive usage of rotation and reflection

The fact is that any finite combination of rotations and reflections is equal to one of these 4 basic operations:

i. no-op (identity transformation, i.e. no actual change happened)
ii. horizontal reflection ⿾
iii. 180° rotation ⿿
iv. vertical reflection (⿾⿿, also ⿿⿾). So this can be treated as one single operator, and unless otherwise specified, the $\boxed{\updownarrow}$ appeared in this article refers to this basic single operation.

The point is that the rotation and the reflection operator can be seen as *linear transformation*, and can be denoted as matrix:

$$⿿: \begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix}, ⿾: \begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}$$

And their combination follows the rules of *matrix multiplication*. So the result must be one of the four:

$$\text{i. no op: } \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \text{ ii. horizontal reflection: } \begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}$$

$$\text{iii. 180° rotation: } \begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix} \text{ iv. vertical reflection: } \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}.$$

And this can be dealt with by algorithm easily. For example:

⿾⿿⿾⿿⿾⿿⿿⿾⿾⿿ =

$$\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix}\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix}\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix}\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix}\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$$

= $\boxed{\updownarrow}$ (i.e. ⿾⿿).

As for the “idempotent property” mentioned by SAT, it seems that it’s more accurate to call the rotation/reflection operations “***involutory***” (i.e. $\mathbf{A^2=I}$**[dentity]**) instead of “*idempotent*” (i.e. $\mathbf{A^2=A}$). SAT argued that this will cause ambiguity (i.e. ⿾⿾ = no-op) and will cause the infinite loop of computer programs. In fact I'm little confused about this since the IDS sequences are always finite, and any computer programs that parse the IDS sequence will only need to reduce the possible redundant unary operations instead of expanding them recursively (e.g. $\boxed{\leftrightarrow}$ = $\boxed{\leftrightarrow}\boxed{\leftrightarrow}$ = $\boxed{\leftrightarrow}\boxed{\leftrightarrow}\boxed{\leftrightarrow}\boxed{\leftrightarrow}$...). Does SAT mean the program that *generate* (instead of parsing) the IDS automatically? If so, as mentioned above, any combination of reflection/rotation operator in an IDS representation will be reduced to **only one** basic operation (denote as "BO-OP" below). That means when generating IDS sequence, if we need to perform the BO-OP, we only need to perform once. So when called the function with BO-OP flag *on*, the next recursive call that pushed into the call stack must **do not need to be** the BO-OP operation, so the BO-OP flag of this call **must** be *off*. This can be easily controlled when programming. Thus we can avoid computer program calling BO-OP recursively (e.g. $\boxed{\leftrightarrow}\boxed{\leftrightarrow}\boxed{\leftrightarrow}\boxed{\leftrightarrow}$...), and avoid the stalemate.

2) Using sequences as argument of rotation/reflection

SAT also argues that using sequences as argument of rotation/reflection operator will cause ambiguous/strange expressions of IDS with an example: $\boxed{\leftrightarrow}$⿰$\boxed{\curvearrowright}$干片, and appeal to ban using sequences as arguments of rotation/reflection operator. But in fact, it's *at least theoretically possible* to design an algorithm that, for example, converted the IDS $\boxed{\leftrightarrow}$⿰$\boxed{\curvearrowright}$干片 mentioned above to it's canonical form: ⿰爿士, and below is the analysis:

An IDS sequence can be seen as a *prefix expression* (i.e. *Polish notation*), with the IDCs as operators, and the ideographs/strokes/components as operands, which is easy to be dealt by computer. What's more important, the rotation/reflection operator satisfies *distributive law* or "*quasi-distributive law*" with traditional IDCs, and here is the explanation (For convenience and easy to be understood, below I'll also illustrate the law with infix expression, which is easy to be read and understood by human, and in this kind of expressions, traditional IDC will be denoted as Cx, in which x is the last hex digit of the code point of the IDC (i.e. U+2FFx), instead of the IDC itself, to avoid confusing with prefix expression. For example, C0 denotes ⿰, C1 denotes ⿱, and CB denotes ⿻, and the rotation is denoted as R0, the reflection is denoted as R1):

$R1(op1\ C1\ op2) = R1(op1)\ C1\ R1(op2)$, [①] i.e. $\boxed{\leftrightarrow}$⿱op1 op2 = ⿱$\boxed{\leftrightarrow}$op1$\boxed{\leftrightarrow}$ op2

$R1(op1\ C0\ op2) = R1(op2)\ C0\ R1(op1)$, [②] i.e. $\boxed{\leftrightarrow}$⿰op1 op2 = ⿰$\boxed{\leftrightarrow}$op2$\boxed{\leftrightarrow}$ op1

$R0(op1\ C5\ op2) = R0(op1)\ C6\ R0(op2)$, i.e. $\boxed{\curvearrowright}$⿵op1 op2 = ⿶$\boxed{\curvearrowright}$op1$\boxed{\curvearrowright}$ op2

$R0(op1\ C9\ op2) = R0(op1)\ CA\ R0(op2)$, i.e. $\boxed{\curvearrowright}$⿹op1 op2 = ⿺$\boxed{\curvearrowright}$op1$\boxed{\curvearrowright}$ op2

Anyway, every Rx operator satisfies such laws with every Cx operator, the results mentioned above are only some typical examples, and I'll not pinpoint every result here. These laws can be easily implemented in recursive steps of parsing IDSes of computer algorithms, and, what's more important, when these laws was implemented, **every operand of Rx operator will be single character/stroke/component instead of sequence when recursion ended**. And that's just one step away from the canonical form that we want. For example, following the laws mentioned above, $\boxed{\leftrightarrow}$⿰$\boxed{\curvearrowright}$干片 can be converted to ⿰$\boxed{\leftrightarrow}$片$\boxed{\leftrightarrow}\boxed{\curvearrowright}$干. Obviously, if we link $\boxed{\leftrightarrow}$片 with 爿, $\boxed{\leftrightarrow}\boxed{\curvearrowright}$干 (i.e $\boxed{\updownarrow}$干) with 士, it's clear that we will get the final canonical form ⿰爿士. To achieve this goal, what we need is simply a database that link the possible relations of reflection/rotation of characters, since the number of encoded ideographs is finite. For example, a given ideograph A, is the **i) horizontal reflection of A ii) rotation of A iii) vertical reflection of A** A itself, or another encoded

[①] That's just like a(b+c)=ab+ac, the so-called distributive law.

[②] Note in this situation the expression does NOT satisfies the distributive law, as the positions of op1 and op2 were reversed, and the IDC is not commutative operator, so I called the situation "quasi-distributive".

ideograph? Or does some two of them not encoded, but identical due to the symmetry property of this ideograph? Creating such a database may require extensive human work, however, with proper image processing tool and data as aid, it's unlikely that many mistakes will be made. So this database keeps 3 entries for every ideograph as mentioned above, and can be efficiently indexed by computer algorithms (at most ~300000 entries, which is not a large number), which means this process is not too compute-intensive at all.

And by the way, an interesting phenomenon is that when applied with the law mentioned above, we may not need proposed IDC **IDEOGRAPHIC DESCRIPTION CHARACTER SURROUND FROM RIGHT** and **IDEOGRAPHIC DESCRIPTION CHARACTER SURROUND FROM RIGHT (This does NOT mean that the author think these IDCs shouldn't be encoded)**, even already encoded U+2FF6 ⿶, U+2FF9 ⿹ and U+2FFA ⿺ at all, since, for example, any ⿺op1 op2 can be represented as ⿾⿹⿾op1⿾op2. However, this will allow sequences as arguments of reflection/rotation operator without fully converted, and this will possibly cause computer algorithms hard to find the relations, which will cause utter chaos. So this is just a random thought, and can be ignored.

2. Subtraction (㇯)

There are also some concerns regarding the ambiguity of subtraction operator ㇯ when the subtrahend was arbitrary ideographs. Since this operator is somewhat arbitrary, this seems reasonable that the subtrahend should be restricted to minor strokes. Based on this, however, I argue that other ambiguity issues are either resolvable or unimportant:

First is the order of subtracted strokes (㇯ ㇯豕丿㇏ and ㇯ ㇯豕㇏丿), this can be done by reordering the subtracted strokes when multiple consecutive subtraction operators are used, and can be done easily by computer algorithms.

As for sequences used as subtrahend (e.g. ㇯豕⿱丿㇏ vs ㇯㇯豕丿㇏), it seems not matter what the structure of subtrahend is, that means, we may try complete drop the traditional IDCs appeared in subtrahend, and adjust the number of subtraction operators. That can be done after a subtrahend finished converting to its canonical form, we count the total operators (denoted as k) of the subtrahend sequence, drop all traditional binary and ternary IDSs, and added k-1 subtraction symbols. This is also not hard to be implemented by computer algorithms. However, this may cause different characters corresponded to the same IDS(for example, if ㇯A⿱BC and ㇯A⿰BC is not the same ideograph), which will return false positives. However, this may be harmless. Also, the converted results by this procedure may be hard to be understood by human, but this seems also harmless since this was mainly done for machine checking.

Another concern is that since subtraction operator has no inverse operator, if there are also subtraction operators in subtrahend, thus will cause problems when using methods above. For example, if we represent 其 as ㇯其㇯八丿, there will be no simple way to convert this form to its canonical form. A worse example would be ㇯A㇯⿰BCD, when applied the method mentioned above, the result would be ㇯ ㇯A㇯BCD, which is an incorrect result. However, since the subtrahend is usually single stroke or combination of strokes, it seems no reason that the subtrahend needs to use this operator ㇯ to describe itself, so maybe we can simply prohibit the usage of the subtraction operator in the subtrahend.

The last problem is the 大-太-犬 problem mentioned by SAT. However, I argue that actually this situation may be rare. The ideographs that needed to use the subtraction operator typically has strong connection with some encoded character (for example, 共亥其巨且 and ㇯東丿), it's easy to identify that what will we get if we add a stroke to them, and is unlike to the situation such as

"人 add a stroke we get 及". Even the "大-太-犬 situation" occurred, the structure and stroke count of involved ideograph usually quite simple and less, which is easy to be identified by human. So this issue may be not a big problem.

(End of Document)