Automated Data Transformation with Domain-Adaptive Inductive Programming (Supplemental Material)

Lidia Contreras-Ochando[1], Cèsar Ferri[1], José Hernández-Orallo[1], Fernando Martínez-Plumed[1], María José Ramírez-Quintana[1], Susumu Katayama[2]
[1] Universitat Politècnica de València, Spain
[2] University of Miyazaki, Japan
{liconoc,cferri,jorallo,fmartinez,mramirez}@dsic.upv.es,
skata@cs.miyazaki-u.ac.jp

A. Domain Specific Induction using MagicHaskeller

MagicHaskeller [Katayama2012] works in two steps: *(1)* The *Hypotheses Generation* phase, and *(2)* the *Hypotheses Selection* phase. In (1), *MagicHaskeller* starts with a predefined d_{max} value (maximum d allowed for the solution) and a set of b functions in the library. Then, *MagicHaskeller* continues with the preparation of hypotheses by generating all the type-correct expressions that can be expressed by function application and lambda abstraction using up to the maximum depth (d_{max}) the functions provided in the library. Although *MagicHaskeller* is very powerful for finding the simplest and most effective solutions (that is, those with smallest Kolmogorov complexity), depending on the problem, the solution might require the combination of many function symbols (that is, a solution with a large depth d). When the d required is higher than the d_{max} value used, *MagicHaskeller* is not able to find the solution (because it cannot reach the necessary number of functions combined). Trying to increase the d_{max} value to achieve the result may cause an increment of time over the top. On the contrary, trying to reduce d, we may be tempted to add many powerful and abstract functions to the library. But, in this case, *MagicHaskeller* will have too many primitives to choose from (the breadth value b), and may not find it either because of the time needed to combine all of them.

Figure 1 (a) illustrates the time used by *MagicHaskeller* in this phase when we vary both the number of functions included in the library (b) and the maximum depth value to obtain the solution (d_{max}).

Finally, in phase (2) we can provide one or more examples (as I/O pairs) to solve a specific problem. *MagicHaskeller* will use the combinations learnt at (1) to find one or more possible solutions to the problem. This solution (if exists) will be a combination of d functions (where $d \leq d_{max}$). In this regard, Figure 1(b) shows the time

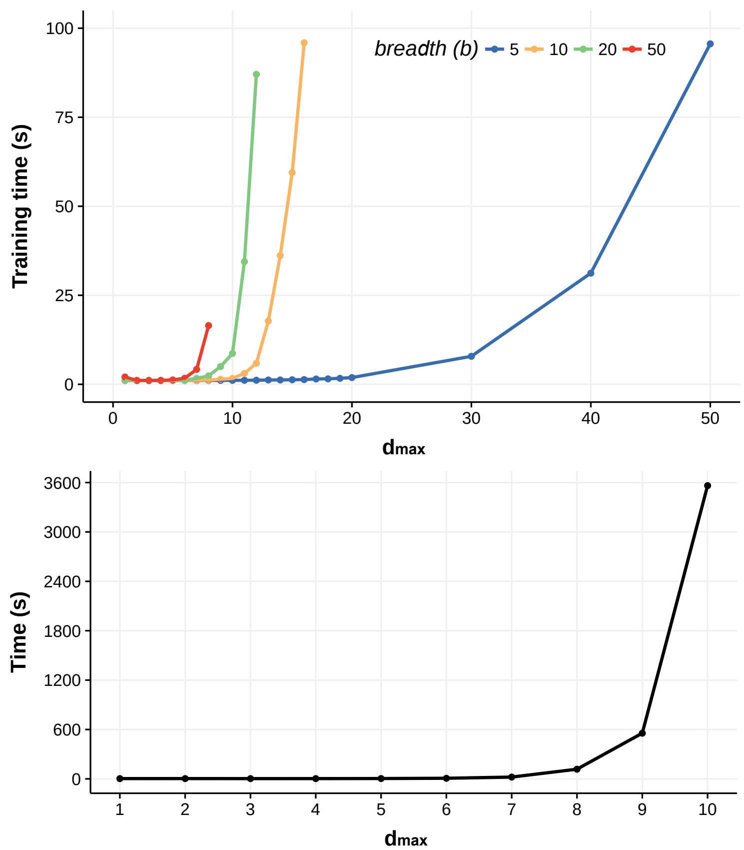

Figure 1: **(a)** Time *MagicHaskeller* needs for training with a set of primitives depending on the maximum number of primitives that are allowed in any synthesised function (d_{max}) and the number of primitives in the set (b). **(b)** Time *MagicHaskeller* needs for training and solve the same problem (concatenate two strings), using a set fixed of $b = 15$ primitives, with varying d_{max} from 1 to 10.

spent during phases (1) and (2) to solve an specific problem (with actual solution of $d = 1$), using the same set of functions (with $b = 15$), but changing the d_{max} value. We acknowledge that d_{max} value has a strong influence too even when there are solutions that require fewer primitives than the maximum depth. Given the heuristics and optimisations included in *MagicHaskeller*, it is still possible to have solutions in cases where $O(b^d)$ grows very fast, but we still see the exponential behaviour in both cases. In the next sections we will show that a good trade-off between d and b can be achieved by using specific domain libraries. Thus, in that follows, we will refer to our approach as *Domain-Specific Induction (DSI)*.

A.1 List of functions/primitives

A.1.1 Default BK: MagicHaskeller's library

By default, *MagicHaskeller* includes a list of 189 basic Haskell functions:

Id	Function
001	0 :: Int
002	1 :: Int
003	(++) :: forall a . (->) ([a]) ([a] -> [a])
004	filter :: forall a . (a -> Bool) -> [a] -> [a]
005	negate :: Ratio Int -> Ratio Int
006	abs :: Ratio Int -> Ratio Int
007	sum :: (->) ([Ratio Int]) (Ratio Int)
008	product :: (->) ([Ratio Int]) (Ratio Int)
009	(+) :: Ratio Int -> Ratio Int -> Ratio Int
010	(-) :: Ratio Int -> Ratio Int -> Ratio Int
011	(*) :: Ratio Int -> Ratio Int -> Ratio Int
012	(/) :: Ratio Int -> Ratio Int -> Ratio Int
013	fromIntegral :: Int -> Ratio Int
014	properFraction :: (->) (Ratio Int) ((Int, Ratio Int))
015	round :: (->) (Ratio Int) Int
016	floor :: (->) (Ratio Int) Int
017	ceiling :: (->) (Ratio Int) Int
018	(i :: Ratio Int -> Int -> Ratio Int
019	(%) :: Int -> Int -> Ratio Int
020	numerator :: (->) (Ratio Int) Int
021	denominator :: (->) (Ratio Int) Int
022	[] :: forall a . [a]
023	(:) :: forall a . a -> [a] -> [a]
024	foldr
025	drop 1
026	(+) :: Int -> Int
027	n x f -> iterate f x !! (n::Int)
028	Nothing :: forall a . Maybe a
029	Just :: forall a . a -> Maybe a
030	maybe
031	True :: Bool
032	False :: Bool
033	iF :: forall a . (->) Bool (a -> a -> a)
034	(+) :: (->) Int ((->) Int Int)
035	(&&) :: (->) Bool ((->) Bool Bool)
036	(——) :: (->) Bool ((->) Bool Bool)
037	not :: (->) Bool Bool

Id	Function
038	(-) :: Int -> Int -> Int
039	(*) :: Int -> Int -> Int
040	map
041	concatMap
042	length :: forall a . (->) ([a]) Int
043	replicate :: forall a . Int -> a -> [a]
044	take :: forall a . Int -> [a] -> [a]
045	drop :: forall a . Int -> [a] -> [a]
046	takeWhile :: forall a . (a -> Bool) -> [a] -> [a]
047	dropWhile :: forall a . (a -> Bool) -> [a] -> [a]
048	reverse :: forall a . [a] -> [a]
049	and :: (->) ([Bool]) Bool
050	or :: (->) ([Bool]) Bool
051	any
052	all
053	zipWith
054	null :: forall a . (->) ([a]) Bool
055	abs :: (->) Int Int
056	foldl
057	total head
058	total last
059	total init
060	enumFromTo :: Int -> Int -> [Int]
061	enumFromTo :: Char -> Char -> [Char]
062	fmap :: forall a b . (a -> b) -> (->) (Maybe a) (Maybe b)
063	either
064	gcd :: Int -> Int -> Int
065	lcm :: Int -> Int -> Int
066	sum :: (->) ([Int]) Int
067	product :: (->) ([Int]) Int
068	(==)
069	(/=)
070	compare
071	(<=)
072	(<)
073	max
074	min
075	sortBy :: forall a . (a -> a -> Ordering) -> [a] -> [a]
076	nubBy :: forall a . (a -> a -> Bool) -> [a] -> [a]
077	deleteBy :: forall a . (a -> a -> Bool) -> a -> [a] -> [a]
078	dropWhileEnd :: forall a . (a -> Bool) -> [a] -> [a]

Id	Function
079	transpose :: forall a . [[a]] -> [[a]]
080	toUpper :: (->) Char Char
081	toLower :: (->) Char Char
082	ord :: Char -> Int
083	isControl :: (->) Char Bool
084	isSpace :: (->) Char Bool
085	isLower :: (->) Char Bool
086	isUpper :: (->) Char Bool
087	isAlpha :: (->) Char Bool
088	isAlphaNum :: (->) Char Bool
089	isDigit :: (->) Char Bool
090	isSymbol :: (->) Char Bool
091	isPunctuation :: (->) Char Bool
092	isPrint :: (->) Char Bool
093	10 :: Int
094	20 :: Int
095	30 :: Int
096	40 :: Int
097	' ' :: Char
098	1 :: Double
099	10 :: Double
100	100 :: Double
101	1000 :: Double
102	succ :: Double -> Double
103	negate :: Double -> Double
104	abs :: Double -> Double
105	signum :: Double -> Double
106	recip :: Double -> Double
107	sum :: (->) ([Double]) Double
108	product :: (->) ([Double]) Double
109	(+) :: Double -> Double -> Double
110	(-) :: Double -> Double -> Double
111	(*) :: Double -> Double -> Double
112	(/) :: Double -> Double -> Double
113	fromIntegral :: Int -> Double
114	properFraction :: (->) Double ((Int, Double))
115	round :: (->) Double Int
116	floor :: (->) Double Int
117	ceiling :: (->) Double Int
118	truncate :: (->) Double Int
119	(i :: Double -> Int -> Double

Id	Function
120	pi :: Double
121	lines :: [Char] -> [[Char]]
122	words :: [Char] -> [[Char]]
123	unlines :: [[Char]] -> [Char]
124	unwords :: [[Char]] -> [Char]
125	scanl :: forall a b . (a -> b -> a) -> a -> [b] -> [a]
126	scanr :: forall a b . (a -> b -> b) -> b -> [a] -> [b]
127	scanl:: forall a . (a -> a -> a) -> [a] -> [a]
128	scanr:: forall a . (a -> a -> a) -> [a] -> [a]
129	show :: Int -> [Char]
130	(,) :: forall a b . a -> b -> (a, b)
131	uncurry
132	elem
133	nub
134	find :: forall a . (a -> Bool) -> [a] -> Maybe a
135	findIndex
136	findIndices
137	deleteFirstsBy :: forall a . (a -> a -> Bool) -> [a] -> [a] -> [a]
138	unionBy :: forall a . (a -> a -> Bool) -> (->) ([a]) ([a] -> [a])
139	intersectBy :: forall a . (a -> a -> Bool) -> (->) ([a]) ([a] -> [a])
140	insertBy :: forall a . (a -> a -> Ordering) -> a -> [a] -> [a]
141	isOctDigit :: (->) Char Bool
142	isHexDigit :: (->) Char Bool
143	catMaybes :: forall a . [Maybe a] -> [a]
144	listToMaybe :: forall a . (->) ([a]) (Maybe a)
145	maybeToList :: forall a . (->) (Maybe a) ([a])
146	exp :: Double -> Double
147	log :: Double -> Double
148	sqrt :: Double -> Double
149	(**) :: Double -> Double -> Double
150	logBase :: Double -> Double -> Double
151	sin :: Double -> Double
152	cos :: Double -> Double
153	tan :: Double -> Double
154	asin :: Double -> Double
155	acos :: Double -> Double
156	atan :: Double -> Double
157	sinh :: Double -> Double
158	cosh :: Double -> Double
159	tanh :: Double -> Double
160	asinh :: Double -> Double

Id	Function
161	acosh :: Double -> Double
162	atanh :: Double -> Double
163	floatDigits :: Double -> Int
164	exponent :: Double -> Int
165	significand :: Double -> Double
166	scaleFloat :: Int -> Double -> Double
167	atan2 :: Double -> Double -> Double
168	(,,) :: forall a b c . a -> b -> c -> (a, b, c)
169	Left :: forall a b . a -> Either a b
170	Right :: forall b a . b -> Either a b
171	zip :: forall a b . (->) ([a]) ((->) ([b]) ([(a, b)]))
172	zip3 :: forall a b c . (->) ([a]) ((->) ([b]) ((->) ([c]) ([(a, b, c)])))
173	unzip :: forall a b . (->) ([(a, b)]) (([a], [b]))
174	unzip3 :: forall a b c . (->) ([(a, b, c)]) (([a], [b], [c]))
175	odd :: Int -> Bool
176	even :: Int -> Bool
177	isPrefixOf
178	isSuffixOf
179	isInfixOf
180	stripPrefix
181	lookup
182	sort
183	intersperse :: forall a . a -> [a] -> [a]
184	subsequences :: forall a . [a] -> [[a]]
185	permutations :: forall a . [a] -> [[a]]
186	inits :: forall a . [a] -> [[a]]
187	tails :: forall a . [a] -> [[a]]
188	mapAccumL
189	mapAccumR

A.1.2 Freetext: Basic string manipulation functions

Although *MagicHaskeller* is able to solve many string and boolean problems by using its default library, this list of functions is not enough to solve more complex problems. For instance, the transformation of the date '29-03-86' into '29/03/86' is impossible to solve with *MagicHaskeller*'s default library since there is a need to replace each dash symbol ('-') with a slash symbol ('/'), and *MagicHaskeller* is unable to generate or use any character or digit if it is not defined as constant in its library or if it is not provided as an input parameter.

Following this and some other examples [Nishida2016] and the most common operators used by other data science tools [noaa][Kandel et al.2011][noab], we have added to *MagicHaskeller* many new functions for solving common problems related to string manipulation:

- **Constants:** Symbols, numbers, words or list of words.

- **Map:** Boolean functions for checking string structures.
- **Transform:** Functions that return the string transformed using one or more of the following operations:
  - **Add:** Appending elements to a string, adding them at the beginning, ending or a fixed position.
  - **Split:** Splitting the string into two or more strings by positions, constants or a given parameter.
  - **Concatenate:** Joining strings, elements of an array, constants or given parameters with or without adding other parameters or constants between them.
  - **Replace:** Changing one or more string elements by some other given element . This operation includes converting a string to uppercase and lowercase.
  - **Exchange:** Swapping elements inside strings.
  - **Delete/Drop/Reduce:** Deleting one or more string elements by some other given parameter, a position, size or mapping some parameter or constant.
  - **Extraction:** Get one or more string elements.

Concretely, we have added 108 functions to solve the string operations:

Id	Function
001	2 :: Int
002	3 :: Int
003	4 :: Int
004	5 :: Int
005	6 :: Int
006	7 :: Int
007	8 :: Int
008	9 :: Int
009	11::Int
010	12::Int
011	13::Int
012	14::Int
013	15::Int
014	16::Int
015	17::Int
016	18::Int
017	19::Int
018	21::Int
019	22::Int
020	23::Int

Id	Function
021	24::Int
022	25::Int
023	26::Int
024	27::Int
025	28::Int
026	29::Int
027	31::Int
028	1900::Int
029	2000::Int
030	dash :: [Char]
031	slash :: [Char]
032	dot :: [Char]
033	comma :: [Char]
034	colon :: [Char]
035	lBracket :: [Char]
036	rBracket :: [Char]
037	at :: [Char]
038	hash :: [Char]
039	lparentheses :: [Char]
040	rparentheses :: [Char]
041	space :: [Char]
042	zero :: [Char]
043	nineteen :: [Char]
044	twenty :: [Char]
045	firstElement :: [Char]
046	middleElement :: [Char]
047	lastElement :: [Char]
048	addPunctuationString :: [Char] -> [Char] -> [[Char]]
049	splitStringWithoutPunctuation :: [Char] -> [[Char]]
050	setPunctuationArray :: [[Char]] -> [Char] -> [[Char]]
051	changePunctuationArray :: [[Char]] -> [Char] -> [[Char]]
052	changePunctuationString :: [Char] -> [Char] -> [Char]
053	deletePunctuationArray :: [[Char]] -> [[Char]]
054	deletePunctuationString :: [Char] -> [Char]
055	deleteSomePunctuationString :: [Char] -> [Char] -> [Char]
056	splitStringByPunctuation :: [Char] -> [Char] -> [[Char]]
057	splitStringWithPunctuation :: [Char] -> [[Char]]
058	splitStringTakeOffPunctuation :: [Char] -> [[Char]]
059	swapElementsString :: Int -> Int -> [Char] -> [Char]
060	swapElementsArray :: Int -> Int -> [[Char]] -> [[Char]]
061	appendPositionArray :: [[Char]] -> [Char] -> Int -> [[Char]]

Id	Function
062	appendPositionString :: [Char] -> [Char] -> Int -> [Char]
063	appendNextToLast :: [[Char]] -> [Char] -> [[Char]]
064	append :: [Char] -> [Char] -> [Char]
065	append_first :: [[Char]] -> [Char] -> [[Char]]
066	append_middle :: [[Char]] -> [Char] -> [[Char]]
067	append_last :: [[Char]] -> [Char] -> [[Char]]
068	prepend :: [Char] -> [Char] -> [Char]
069	prepend_first :: [[Char]] -> [Char] -> [[Char]]
070	prepend_middle :: [[Char]] -> [Char] -> [[Char]]
071	prepend_last :: [[Char]] -> [Char] -> [[Char]]
072	replacePositionArray :: [[Char]] -> [Char] -> Int -> [[Char]]
073	replacePositionString :: [Char] -> [Char] -> Int -> [Char]
074	replacePositionArrayFixedSize :: [[Char]] -> [Char] -> [Char] -> [[Char]]
075	replaceAll :: [Char] -> [Char] -> [Char] -> [Char]
076	replaceNextToLast :: [[Char]] -> [Char] -> [[Char]]
077	toLowString :: [Char] -> [Char]
078	toUpperString :: [Char] -> [Char]
079	reduceWord :: [Char] -> Int -> [Char]
080	takeOneOfArray :: [[Char]] -> Int -> [Char]
081	takeOneOfFixedSizeArray :: [[Char]] -> [Char] -> [Char]
082	takeOneOfFixedSizeString :: [Char] -> [Char] -> [Char]
083	joinStringsWithPunctuation :: [Char] -> [Char] -> [Char] -> [Char]
084	getOneWordByPosition :: [Char] -> Int -> [Char]
085	getFirstWord :: [Char] -> [Char]
086	getLastWord :: [Char] -> [Char]
087	getOneCharacterByPosition :: [Char] -> Int -> [Char]
088	getFirstCharacter :: [Char] -> [Char]
089	getLastCharacter :: [Char] -> [Char]
090	getStartToFirstSymbolOccurrence :: [Char] -> [Char] -> [Char]
091	getStartToLastSymbolOccurrence :: [Char] -> [Char] -> [Char]
092	getLastSymbolOccurrenceToEnd :: [Char] -> [Char] -> [Char]
093	getFirstSymbolOccurrenceToEnd :: [Char] -> [Char] -> [Char]
094	joinArrayWithPunctuation :: [[Char]] -> [Char] -> [Char]
095	joinStringsWithoutPunctuation :: [Char] -> [Char] -> [Char]
096	setParentheses :: [Char] -> [Char]
097	getCaps :: [Char] -> [Char]
098	reduceSpaces :: [Char] -> [Char]
099	setBrackets :: [Char] -> [Char]
100	completeBrackets :: [Char] -> [Char]
101	completeParentheses :: [Char] -> [Char]
102	getFirstDigitToEnd :: [Char] -> [Char]

Id	Function
103	getStartToFirstDigit :: [Char] -> [Char]
104	insert_first :: [[Char]] -> [Char] -> [[Char]]
105	insert_last :: [[Char]] -> [Char] -> [[Char]]
106	deleteParentheses :: [Char] -> [Char]
107	changeSomePunctuationString :: [Char] -> [Char] -> [Char] -> [Char]
108	removeWords :: [Char] -> [[Char]] -> [Char]

A.1.3 Domain functions

With this set of functions in the system's library, we are able to solve many common string manipulation problems. However, when data belong to a particular domain and the problem at hand ends up being a very exclusive task pertaining to that domain, more precise functions are needed in order to get correct results considering the context. The following domains and their functions are added to the library.

A.1.3.1 Dates Domain

Id	Function
01	getDayCardinalString::[Char]->[Char]
02	getDayCardinalArray::[[Char]]->[Char]
03	getDayOrdinal::[Char]->[Char]
04	getWeekDayArray::[[Char]]->[Char]
05	getWeekDayString::[Char]->[Char]
06	getMonthNameString::[Char]->[Char]
07	getMonthNameArray::[[Char]]->[Char]
08	convertMonth::[Char]->[Char]
09	getYearString::[Char]->[Char]
10	getYearArray::[[Char]]->[Char]
11	convertMonthToNumeric::[Char]->[Char]
12	convertMonthToString::[Char]->[Char]
13	takeTwoOfThreeArray::[[Char]]->Int->Int->[[Char]]
14	getMonthArray::[[Char]]->[Char]
15	getMonthString::[Char]->[Char]
16	convertMonthToNumericWithinArray::[[Char]]->[[Char]]
17	convertMonthToStringWithinArray::[[Char]]->[[Char]]
18	reduceMonthWithinArray::[[Char]]->[[Char]]
19	changeDateFormat::[Char]->[Char]->[[Char]]
20	convertDayOrdinalWithinArray::[[Char]]->[[Char]]
21	reduceYear::[Char]->[Char]
22	reduceYearWithinArray::[[Char]]->[[Char]]
23	reduceMonth::[Char]->[Char]

A.1.3.2 Emails Domain

Id	Function
1	getWordsBeforeAt :: [Char] -> [Char]
2	getWordsAfterAt :: [Char] -> [Char]
3	getWordsBeforeDot :: [Char] -> [Char]
4	getWordsAfterDot :: [Char] -> [Char]
5	getWordsBetweenAtAndDot :: [Char] -> [Char]
6	appendAt :: [Char] -> [Char]
7	prependAt :: [Char] -> [Char]
8	joinStringsWithAt :: [Char] -> [Char] -> [Char]
9	dotcom :: [Char]

A.1.3.3 Names Domain

Id	Function
01	addMaleNomenclature :: [Char] -> Int -> [Char]
02	addFemaleNomenclature :: [Char] -> Int -> [Char]
03	deleteNomenclature :: [Char] -> [Char]
04	getNomenclature :: [Char] -> [Char]
05	reduceNameSecondWord :: [Char] -> [Char] -> [Char]
06	getGenderByNomenclature :: [Char] -> [Char]
07	deleteNomenclatureAndPunctuation :: [Char] -> [Char]
08	reduceNamesFirstPlace :: [Char] -> [Char]
09	reduceNameFirstPlace :: [Char] -> [Char]
10	reduceNameWithSurnameSecondPlace :: [Char] -> [Char]
11	reduceNameWithSurnamesSecondPlace :: [Char] -> [Char]
12	initialsNameFirstPlace :: [Char] -> [Char]

A.1.3.4 Phones Domain

Id	Function
1	addPhonePrefix :: [Char] -> Int -> [Char]
2	addPhonePrefixByCountry :: [Char] -> [Char] -> [Char]
3	addPhonePrefixByCountryCode :: [Char] -> [Char] -> [Char]
4	addPlusInPrefix :: [Char] -> [Char]
5	getPhoneNumber :: [Char] -> [Char]

A.1.3.5 Times Domain

Id	Function
01	integerToTime :: Int -> [Char]
02	appendTimeElement :: [Char] -> Int -> [Char]
03	convertTimeByTimeZone :: [Char] -> [Char] -> [Char] -> [Char]
04	getTime :: [Char] -> [Char]
05	getHour :: [Char] -> [Char]
06	getMinutes :: [Char] -> [Char]
07	getSeconds :: [Char] -> [Char]
08	appendOclockTime :: [Char] -> [Char]

Id	Function
09	appendSomeTime :: [Char] -> Int -> [Char]
10	changeHour :: [Char] -> Int -> [Char]
11	changeMinutes :: [Char] -> Int -> [Char]
12	changeSeconds :: [Char] -> Int -> [Char]
13	deleteLastTimePosition :: [Char] -> [Char]
14	increaseHour :: [Char] -> Int -> [Char]
15	decreaseHour :: [Char] -> Int -> [Char]
16	increaseMinutes :: [Char] -> Int -> [Char]
17	decreaseMinutes :: [Char] -> Int -> [Char]
18	increaseSeconds :: [Char] -> Int -> [Char]
19	decreaseSeconds :: [Char] -> Int -> [Char]
20	convertTimeTo24hoursFormat :: [Char] -> [Char]
21	convertTimeTo12hoursFormat :: [Char] -> [Char]
22	convertTimeFormat :: [Char] -> [Char] -> [Char]
23	get12hoursFormatAuxiliar :: [Char] -> [Char]
24	delete12hoursFormatAuxiliar :: [Char] -> [Char]

A.1.3.6 Units Domain

Id	Function
1	unitsConversion :: Float -> [Char] -> [Char] -> Float
2	getUnits :: [Char] -> [Char]
3	getSystem :: [Char] -> [Char]
4	setUnits :: Float -> [Char] -> [Char]

B. Describing the problems using meta-features

In order to describe different characteristics of the inputs and use them with the domain classifier and function ranker, we have defined the following descriptive *meta-features* that can be extracted automatically:

Id	Meta-feature
01	end_digit
02	end_dotAndWord
03	end_lower
04	end_upper
05	has_1at
06	has_1comma
07	has_2blank
08	has_2colon
09	has_2dash
10	has_2digits
11	has_2dot
12	has_2slash

Id	Meta-feature
13	has_4digits
14	has_6digits
15	has_8digits
16	has_9OrMoredigits
17	has_at
18	has_blanks
19	has_capitalAndDot
20	has_colon
21	has_courtesyTitles
22	has_dash
23	has_dayName
24	has_dot
25	has_hourStructure
26	has_hoursWords
27	has_lettersAndDot
28	has_lowers
29	has_monthName
30	has_numbers
31	has_numbersInsideParenthesis
32	has_only1Word
33	has_only2Words
34	has_only3Words
35	has_onlyNumbersAndSymbols
36	has_ordinalNumbers
37	has_plus
38	has_punctuation
39	has_slash
40	has_unitsSystem
41	has_uppers
42	has_wordAndComma
43	has_wordsJointByDash
44	is_emailStructure
45	is_empty
46	is_LongDateStructure
47	is_NA
48	is_onlyAlphabetic
49	is_onlyNumeric
50	is_onlyPunctuation
51	is_shortDateStructure
52	start_digit
53	start_lower

Id	Meta-feature
54	start_upper

C. Data

Table 1 shows the description of the 95 datasets included in the experiments.

id	Description	Expected Output
1, 2	Add punctuation	The date in numeric format split by a punctuation sign
3 . . . 5	Change format	The date in one particular format
6, 7	Change Punctuation	The date in one particular format
8 . . . 10	Get Day	The day in numeric format
11, 12	Get Day Ordinal	The day in numeric ordinal format
13, 14	Get Month Name	The name of the month
15, 16	Get Week Day	The name of the weekday
17, 18	Reduce Month Name	The name of the month reduced to three letters
19, 20	Set Format	The date split with a punctuation sign and reordered in DMY format
21 . . . 23	Generate Email	An email account created with the name and the domain
24 . . . 27	Get After At	Everything after the at symbol
28, 29	Get Domain	The domain before the dot
30	Before At	Everything before the at symbol
31, 32	Add Title	The name with a title
33, 34	Get Title	The title attached to the name, if exists
35, 36	Generate Login	A login generated using the name
37 . . . 45	Reduce name	The name reduced before the surname(s)
46 . . . 50	Add Prefix by Country	The list of phone numbers in a unique format with the prefix of the countries
51, 52	Delete Parentheses	The list of phone numbers without parentheses
53, 54	Get Number	A phone number presented in the string, if exists
55 . . . 59	Set Prefix	The list of phone numbers in a unique format with the prefix
60, 61	Set Punctuation	A phone number split by a punctuation sign
62, 63	Add Time	The time increasing the hour by the integer
64, 65	Append o'clock Time	The time appending an o'clock time, if possible
66, 67	Append Time	The time appending the integer as new component, if possible
68, 69	Convert Time	The time formatted to 24 hours format
70, 71	Convert Time	The time formatted to a given format
72, 73	Convert Time	The time formatted to 12 hours format
74 . . . 77	Convert Time	The time changed from the first time zone to the second
78, 79	Delete Time	The time deleting the last component
80, 81	Get Hour	The hour component
82, 83	Get Minutes	The minutes component
84, 85	Get Time	A time presented in the string
86 . . . 89	Convert Units	The value transformed to a different magnitude
90, 91	Get System	The system represented by the magnitude
92, 93	Get Units	The units of the system
94, 95	Get Value	The numeric value without any magnitude

Table 1: Datasets included in the new data wrangling repository offered for the data mining research community.

D. Extended Results

D.1 Domain classifier

In order to build the domain classifier, we used the 54 descriptive meta-features and off-the-shelf machine learning methods: random forest (from R package `caret`[1]) produced the best results. Table 2 shows the results for the four classifiers tested.

Method	Acc.	Kappa
C5.0 Tree	0.822	0.786
Neural Network	0.741	0.689
Naïve Bayes	0.458	0.350
Random Forest	**0.886**	**0.847**

Table 2: Results for the domain detection using the meta-features with different machine learning methods. The best results are highlighted in bold.

D.2 Function Comparison

We have compared the performance of our approach using the ranking strategy with other DSL data wrangling tools, concretely *Trifacta Wrangler* and *FlashFill*. Flashfill works in the same way as our approach, namely, it uses one or more input instances to try to induce a potential solution which is then applied to the rest of examples. If no solution is found or the problem at hand is not solvable by Flashfill, it returns, respectively, a void function or an error. On the other hand, Trifacta Wrangler works in a slightly different fashion: it tries to discover patterns and perform actions in the entire dataset. Each of these actions can involve one change (e.g.: merge two columns) and they are saved in a final 'recipe'.

It should be noted that, as we have used a d_{max} value equal to 4 in MagicHaskeller, the obtained solutions can concatenate up to 4 functions or constants. Since we want to compare the results in similar conditions, we assume that the number of actions which can be used by Trifacta Wrangler to obtain a solution is similar to the d_{max} value in MagicHaskeller. Therefore, we limit the maximum number of actions in each Wrangler recipe to 4. Since Trifacta Wrangler uses all the elements in the column, the solution presented here tries to solve, at least, the first example.

D.3 Accuracy

In Table 3 we first compare the results (in terms of 'depth') of the dynamic BK and Trifacta Wrangler. Note that, in both approaches, the actual number of functions or actions needed to solve the problem (d) can be smaller than d_{max}. We can see that Trifacta Wrangler is able to detect some data types or domains, for instance: 'url', 'time', 'phone'. With this predefined formats the tool is capable of solve very domain-specific problems such as get the name of the month or the day in a date, detect an email

[1] `https://cran.r-project.org/web/packages/caret/`

id	*Trifacta Wrangler*	*d*	*Dynamic BK*	*d*
1	extractpatterns type: custom col: input1 on: ‘digit2‘ extractpatterns type: custom col: input1 on: ‘digit2‘ limit: 2 extractpatterns type: custom col: input1 on: ‘digit2‘ limit: 3 merge col: input5,input6,input7 with: ’-’ as: ’column1’	4	joinArrayWithPunctuation (splitStringWithoutPunctuation a) dash	3
3	extractpatterns type: custom col: column2 on: ‘dd‘ extractpatterns type: custom col: column2 on: ‘mm‘ limit: 2 extractpatterns type: custom col: column2 on: ‘yyyy‘ merge col: column1,column5,column7 with: ’’ as: ’column8’	4	concat (addPunctuationString a dash)	3
6	replacepatterns col: column2 with: ’-’ on: ‘delim‘ global: true	1	changePunctuationString a dash	2
8	extractpatterns type: custom col: column2 on: ‘dd‘ limit: 2	1	getDayCardinalString a	1
11	extractpatterns type: custom col: column2 on: ‘dd‘ textformat col: column1 type: suffix text: ’th’	2	getDayOrdinal a	1
13	extractpatterns type: custom col: column2 on: ‘month‘	1	getMonthNameString a	1
15	extractpatterns type: custom col: column2 on: ‘dayofweek‘	1	getWeekDayString a	1
17		-	reduceMonth (convertMonth (getMonthString a))	3
19		-	joinArrayWithPunctuation (changeDateFormat a mdy) slash	4
21	merge col: column2,column3 with: ’@’ as: ’column1’ textformat col: column1 type: suffix text: ’.com’ replacepatterns col: column1 with: ” on: ‘’ global: true textformat col: column1 type: removewhitespace	4	joinStringsWithAt a (append dotcom b)	3
24	extractpatterns type: custom col: column2 on: ‘url‘ start: ‘@‘ end: ‘”‘	1	getWordsAfterAt a	1
28	extractpatterns type: custom col: input1 on: /[”@”].+[.]/ extractpatterns type: custom col: input2 on: ‘alpha+‘	2	getWordsBeforeDot (getWordsAfterAt a)	2
30	extractpatterns type: custom col: column2 on: ‘url‘ start: ‘”‘ end: ‘@‘	1	getWordsBeforeAt a	1
31		-	addNomenclature a b	1
33	extractpatterns type: custom col: column2 on: ‘url‘ start: ‘”‘ end: ‘@‘	1	getNomenclature a	1
35	extractpatterns type: custom col: column2 on: ‘alpha2‘ extractpatterns type: custom col: column2 on: ‘alpha2‘ start: ‘delim‘ end: ‘lower+‘ textformat col: column5,column1 type: lowercase merge col: column1,column5 as: ’column6’	4	loginByNameString a	1
37	extractpatterns type: custom col: column2 on: ‘alpha+‘ limit: 3 extractpatterns type: custom col: column5 on: ‘upper‘ merge col: column6,column7 with: ’, ’ as: ’column8’ textformat col: column8 type: suffix text: ’.’	4	reduceNameWithSurname a	1
46		-	addPhonePrefixByCountry a b	1
51	replacepatterns col: column2 with: ” on: ‘(‘ global: true replacepatterns col: column2 with: ” on: ‘)‘ global: true	2	deleteParentheses a	1
53	extractpatterns type: custom col: column2 on: ‘phone‘	1	getPhoneNumber a	1
55	merge col: input2,input1 with: ’-’ as: ’column1’ replacepatterns col: column1 with: ’-’ on: ’ ’	2	addPhonePrefix a b	1
60	extractpatterns type: custom col: input1 on: ‘digit3‘ extractpatterns type: custom col: input1 on: ‘digit3‘ limit: 2 extractpatterns type: custom col: input1 on: ‘digit4‘ start: ‘digit6‘ end: ‘end‘ merge col: input2,input4,input5 with: ’-’ as: ’column1’	4	joinArrayWithPunctuation (splitStringWithoutPunctuation a) dash	3
62		-	increaseHour a b	1
64	textformat col: column2 type: suffix text: ’:00’	1	appendOclockTime a	1
67	merge col: input1,input2 with: ’:’ as: ’column1’	1	appendSomeTime a b	1
68		-	convertTimeTo24hoursFormat a	1
70		-	convertTimeFormat a b	1
72		-	convertTimeTo12hoursFormat a	1
74		-	convertTimeByTimeZone a b c	1
78	replacepatterns col: column2 with: ” on: ‘:digit2end‘	1	deleteLastTimePosition a	1
80	extractpatterns type: custom col: column2 on: ‘digit+‘	1	getHour a	1
82	extractpatterns type: custom col: input1 on: ‘digit+‘ limit: 2 drop col: input2 action: Drop	2	getMinutes a	1
84	extractpatterns type: custom col: column2 on: ‘time‘	1	getTime a	1
86		-	unitsConversion (getValue a) (getUnits a) b	3
90		-	getSystem a	1
92	extractpatterns type: custom col: column2 on: ‘lower+‘	1	getUnits a	1
94	extractpatterns type: custom col: column2 on: ‘digit2.digit2‘	1	getValue a	1

Table 3: Functions obtained by our approach (Dynamic BK) compared with ‘recipe’ expressions of *Trifacta Wrangler*. d is the ‘depth’ of the solution obtained with a $d_{max} = 4$.

id	*input*	*output*	*FlashFill*	*Trifacta Wrangler*	*Dynamic BK*
1	*290386* 250374 121099	*29-03-86* 25-03-74 12-10-99	 25-03-74 12-10-99	 25-03-74 12-10-99	 25-03-74 12-10-99
		Accuracy:	1	1	1
3	*20040717* 20021015 09292015	*17-07-2004* 15-10-2002 29-09-2015	 17/10/2002 17/20/0929		 15-10-2002 29-09-2015
		Accuracy:	0	0	1
6	*29/03/86* 11.02.96 12/10/99	*29-03-86* 11-02-96 12-10-99	 11.02.96 12-10-99	 11-02-96 12-10-99	 11-02-96 12-10-99
		Accuracy:	0.6	1	1
8	*03/29/86* 74-03-31 25-08-85	*29* 31 25	 3 8	 03 08	 31 25
		Accuracy:	0	0	1
11	*3/29/86* 12/99/13 10 12 69	*29th* 13th 10th	 99th 12th	 th 12th	 13th 10th
		Accuracy:	0	0	1
13	*2 of September of 2010, Monday* 13 November 2008 June 23, 2007	*September* November June	 2008 2007	 November June	 November June
		Accuracy:	0.2	1	1
15	*Sunday, 9 November 2014* 2 of September of 2010, Monday Wednesday, 15 October 2003	*Sunday* Monday Wednesday	 2 of September of 2010 Wednesday	 Monday Wednesday	 Monday Wednesday
		Accuracy:	0.8	1	1
17	*15/02/84* 11/30/2017 28/12/2004	*Feb* Nov Dec	 Feb Feb		 Nov Dec
		Accuracy:	0	0	1

Table 4: Example of the results of our approach (Dynamic BK) compared with *FlashFill* and *Trifacta Wrangler* using datasets of *dates*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn, and used in Wrangler to generate the results. Green colour means correct result; Red colour means incorrect result.

or extract the hour of a time. Although Trifacta Wrangler allows the user to select the type of data used and then it solves input problems according on it, there are some limitations when dealing with different formats. For instance, when using the dataset #8 from Table 1, the type of the data is ‘dates’, but each instance is written in a different format: Trifacta Wrangler is not able to detect that they are all dates and throws an ’invalid types’ error. On the contrary, the dynamic BK is able to use the functions adapted to this domain, regardless of their written format. Another problem here is that Trifacta Wrangler is unable to introduce new characters or constants (such as ’@’, ’th’, ’:00’, etc.), the user can introduce them and prefixes or sufixes and this implies the need of more than four actions to deal with some examples. Furthermore, it also uses very strict predefined formats for different types of data (such as dates or times) which lead to errors when small variations in the input formats occur. On the contrary, our approach faces this sort of problems in a different way by considering constants (such

id	*input*	*output*	*FlashFill*	*Trifacta Wrangler*	*Dynamic BK*
21	*Sophia & domain* Logan & domain Lucas & domain	*Sophia@domain.com* Logan@domain.com Lucas@domain.com	Logan@domain.com Lucas@domain.com	Logan@domain.com Lucas@domain.com	Logan@domain.com Lucas@domain.com
		Accuracy:	1	1	1
24	*Nancy.FreeHafer@fourthcoffee.com* Andrew.Cencici@northwind-traders.com Jan.Kotas@litwareinc.com	*fourthcoffee.com* northwind-traders.com litwareinc.com	northwind-traders.com litwareinc.com	northwind-traders.com litwareinc.com	northwind-traders.com litwareinc.com
		Accuracy:	1	1	1
28	*Nancy.FreeHafer@fourthcoffee.com* Andrew.Cencici@northwind-traders.com Jan.Kotas@litwareinc.com	*fourthcoffee* northwind-traders litwareinc	northwind-traders litwareinc	northwind-traders litwareinc	northwind-traders litwareinc
		Accuracy:	1	1	1
28	*Nancy.FreeHafer@fourthcoffee.com* Andrew.Cencici@northwind-traders.com Jan.Kotas@litwareinc.com	*Nancy.FreeHafer* Andrew.Cencici Jan.Kotas	Andrew.Cencici Jan.Kotas	Andrew.Cencici Jan.Kotas	Andrew.Cencici Jan.Kotas
		Accuracy:	1	1	1

Table 5: Example of the results of our approach (Dynamic BK) compared with *FlashFill* and *Trifacta Wrangler* using datasets of *emails*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn, and used in Wrangler to generate the results. Green colour means correct result; Red colour means incorrect result.

id	*input*	*output*	*FlashFill*	*Trifacta Wrangler*	*Dynamic BK*
33	*Dr. Mark Sipser* Louis Johnson, PhD Robert Mills	*Dr.* PhD	Lou Rob		PhD
		Accuracy:	0.4	0.4	1
35	*Guillermo Filiepatos* Federico A. Fithsakampf Carmen Funcsrentano	*gufi* fefi cafu	fe cafuncs	fefi cafu	fefi cafu
		Accuracy:	0	0.6	1
37	*Dr. Eran Yahav* Prof. Kathleen S. Fisher Ken McMillan, II	*Yahav, E.* Fisher, K. McMillan, K.	Fisher, Kathleen S. II, M.	S, K. II, M.	Fisher, K. McMillan, K.
		Accuracy:	0	0.2	1

Table 6: Example of the results of our approach (Dynamic BK) compared with *FlashFill* and *Trifacta Wrangler* using datasets of *names*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn, and used in Wrangler to generate the results. Green colour means correct result; Red colour means incorrect result.

as '@') as functions, although it needs a higher number of them when there are more than one input . The last and most important problem related with Trifacta is that the user needs to know the lenguage behind the tool or some regular expressions in order to solve more complex examples. In order to overcome this problem, our system is able to solve most of the problems using only one example given by the user, without the need of having any technical knowledge.

Tables 4, 5, 6, 7, 8, 9 show some illustrative outcomes obtained for some datasets as well as the accuracy values for each dataset. The first instance (in italics) for each dataset (*input* column) is the one used for inferring the solution for each tool. For each dataset only the three first instances are shown.

In Table 4 we can see the problem of having different formats in the data of one

id	*input*	*output*	*FlashFill*	*Trifacta Wrangler*	*Dynamic BK*
46	*235-7654 & Taiwan* 17-455-81-39 & Spain 25-437-96-20 & South Korea	*(886) 235-7654* (34) 17-455-81-39 (82) 25-437-96-20	(886) 17-455-81-39 (886) 25-437-96-20		(34) 17-455-81-39 (82) 25-437-96-20
		Accuracy:	0	0	1
48	*785-4210 & MDG* 352-7960 & KWT 846-2730 & AND	*(261) 785-4210* (34) 17-455-81-39 (376) 846-2730	(261) 17-455-81-39 (261) 846-2730		(34) 17-455-81-39 (376) 846-2730
		Accuracy:	0	0	1
51	*(693)-785-4210* (481)-352-7960 (568)-734-2190	*693-785-4210* 481-352-7960 568-734-2190	568-734-2190 568-734-2190	481-352-7960 568-734-2190	481-352-7960 568-734-2190
		Accuracy:	1	1	1
53	*425-457-2130, DJs flock by when MTV ax quiz prog: 18:95* John DOE 3 Data [TS]865-000-0000 - - 453442-00 06-23-2009 17:58-19:29, 425-743-1650	*425-457-2130* 865-000-0000 425-743-1650	John DOE 3 Data [TS]865-000-0000 - - 453442-00 06-23-2009 17:58-19:29	425-743-1650	865-000-0000 425-743-1650
		Accuracy:	0	0.8	1
55	*235 7654 & 425* 745-8139 & 425 437-9620 & 425	*425-235-7654* 425-745-8139 425-437-9620	425-745-8139 425-437-9620	425-745-8139 425-437-9620	425-745-8139 425-437-9620
		Accuracy:	1	1	1
60	*3237087700* 1635879240 1854379620	*323-708-7700* 163-587-9240 185-437-9620	163-587-9240 185-437-9620	163-587-9240 185-437-9620	163-587-9240 185-437-9620
		Accuracy:	1	1	1

Table 7: Example of the results of our approach (Dynamic BK) compared with *FlashFill* and *Trifacta Wrangler* using datasets of *phones*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn, and used in Wrangler to generate the results. Green colour means correct result; Red colour means incorrect result.

column. The datasets related with dates have very differents types of dates as examples. Our system is able to detect the different formats and deal with them. On the contrary, Flashfill and Trifacta Wrangler are incapable of detect the different types of dates to work with them or types of dates different that their predefined ones. For instance, FlashFill can not detect '11.02.96' as a date. On the other side, 6 shows how FlashFill and Trifacta Wrangler fail when they has to deal with people's names.

In Table 5 we observe that the three approaches work well with emails. Our approach and Trifacta Wrangler are able to detect emails. FlashFill for its part is able to work with basic string manipulation functions to deal very well with email problems.

In Tables 7, 8 and 9 we can see that although FlashFill is not able to detect many types of data, is capable of solving some examples by using its DSL based on basic string manipulation problems. Here, we can see some strength and weakness in each tool. It is clear that a DSL is not enough to deal with a high range of problems when they become into a domain-related problems. In the same way, even when Trifacta Wrangler is able to solve more problems than FlashFill detecting the domains, it is important to notice that the user needs a high degree of knowledge about the problem to solve, and the language of the tool. In summary, the results show that our approach is autonomous as it recognises the domain and it is more effective in terms of results.

id	*input*	*output*	*FlashFill*	*Trifacta Wrangler*	*Dynamic BK*
62	*01:34:00 + 5* 01:55 + 5 16:15:12 + 5	*06:34:00* 06:55 21:15:12	 6:55:00 6:15:00		 06:55 21:15:12
		Accuracy:	0	0	1
64	*01:34* 07:05 16:15:12	*01:34:00* 07:05:00 16:15:12	 07:05:00 16:15:12:00	 07:05:00 16:15:12:00	 07:05:00 16:15:12
		Accuracy:	0.8	0.8	1
66	*01:34 + 30* 01:55 + 30 16:15:12 + 30	*01:34:30* 01:55:30 16:15:12	 01:55:30 16:15:12:30	 16:15:12 16:15:12:30	 01:55:30 16:15:12
		Accuracy:	0.6	0.8	1
68	*1:34:00 PM CST* 3:40 AM 07:05:59	*13:34:00* 03:40 07:05:59	 10:40:00 22:05:59	 3:40 07:05:59	 03:40 07:05:59
		Accuracy:	0	0.6	0.8
78	*01:34:00* 01:55 16:15:12	*01:34* 01 16:15	 0:00 16:15	 01 16:15	 01 16:15
		Accuracy:	0.2	1	1
80	*01:55* 03:40 AM 08:40 UTC	*01* 03 08	 03 03	 03 08	 03 08
		Accuracy:	1	1	1
82	*1:34:00 PM CST* 3:40 AM 07:05:59	*34* 40 05	 40 05	 40 05	 40 05
		Accuracy:	1	1	1
84	*1:34:00 PM CST* 3:40 AM 07:05:59	*1:34:00* 3:40 07:05:59	 3:40:00 07:05:59	 3:40 07:05:59	 3:40 07:05:59
		Accuracy:	0.4	1	1

Table 8: Example of the results of our approach (Dynamic BK) compared with *FlashFill* and *Trifacta Wrangler* using datasets of *times*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn, and used in Wrangler to generate the results. Green colour means correct result; Red colour means incorrect result.

References

[Kandel et al.2011] Kandel, S.; Paepcke, A.; Hellerstein, J.; and Heer, J. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 3363–3372. ACM.

[Katayama2012] Katayama, S. 2012. An analytical inductive functional programming system that avoids unintended programs. In *Proceedings of the ACM SIGPLAN 2012 workshop on Partial evaluation and program manipulation*, 43–52. ACM.

[Nishida2016] Nishida, K. 2016. 7 Most Practically Useful Operations When Wrangling with Text Data in R.

id	*input*	*output*	*FlashFill*	*Trifacta Wrangler*	*Dynamic BK*
86	*1441.8mg* $\rightarrow g$	*1.4418001*			
	87 s $\rightarrow ns$	8700000.0	87418001		8700000.0
	1854 dam $\rightarrow dm$	185400.0	18418001		185400.0
		Accuracy:	0	0	1
90	*56.77cl*	*Volume*			
	84kg	Mass	Volume		Mass
	1854 dam	Length	Volume		Length
		Accuracy:	0	0	1
92	*56.77cl*	*cl*			
	84kg	kg	kg	g	kg
	1854 dam	dam	dam	dam	dam
		Accuracy:	1	0.8	1
94	*56.77cl*	*56.77*			
	84kg	84	84		84
	1854 dam	1854	1854		1854
		Accuracy:	1	0.4	1

Table 9: Example of the results of our approach (Dynamic BK) compared with *FlashFill* and *Trifacta Wrangler* using datasets of *units*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn, and used in Wrangler to generate the results. Green colour means correct result; Red colour means incorrect result.

[noaa] Key Features of RapidMiner Studio. https://rapidminer.com/products/studio/feature-list/.

[noab] Wrangle Language - Trifacta Wrangler - Trifacta Documentation.