# From Sentence to Action: Splitting AMR Graphs for Recipe Instructions

**Katharina Stein**[1] **Lucia Donatelli**[2] **Alexander Koller**[1]
[1] Department of Language Science and Technology
Saarland Informatics Campus
Saarland University, Saarbrücken, Germany
[2] Vrije Universiteit Amsterdam
De Boelelaan 1105, 1081 HV Amsterdam, Netherlands
kstein@lst.uni-saarland.de

## Abstract

Accurately interpreting the relationships between actions in a recipe text is essential to successful recipe completion. We explore using Abstract Meaning Representation (AMR) to represent recipe instructions, abstracting away from syntax and sentence structure that may order recipe actions in arbitrary ways. We present an algorithm to split sentence-level AMRs into action-level AMRs for individual cooking steps. Our approach provides an automatic way to derive fine-grained AMR representations of actions in cooking recipes and can be a useful tool for downstream, instructional tasks.

## 1 Introduction

Procedural texts are special kinds of text that serve the purpose of guiding humans through the steps required to accomplish a specific task. Recipe texts are an idiosyncratic kind of procedural texts whose successful execution depends on accurately interpreting which actions need to be carried out in which order and which ingredients and tools are involved in each step. For example, the instruction *"**Turn** the dough **out** onto a surface **dusted** with flour and **knead** briefly until smooth."* presents three actions: (i) dusting a surface; (ii) placing the dough on that surface; and (iii) kneading the dough until smooth. In recipe texts, actions often depend on other actions but there is often flexibility with respect to the overall order in which actions are instructed as some actions can take place in parallel or can be carried out at different stages of the cooking process. For example, dusting the surface could be instructed before preparing the dough.

Recipe texts frequently combine several actions in one sentence and often there are no uniform methods for putting specific actions into the same instruction; different versions of the same recipe may even differ in how equivalent or parallel actions are distributed across sentences.

Tasks such as adapting a recipe to a specific situation or presenting a recipe interactively to a user require the generation of a coherent recipe text that presents actions in a potentially different order and combination. To flexibly generate new versions of a recipe that present actions in an adapted order it is necessary to correctly decompose the recipe into the individual actions.

Previous work on recipe texts proposed identifying cooking actions and objects in recipes and representing the dependency relations between them in domain-specific graph representations with nodes for actions, ingredients and tools (Mori et al., 2014a; Yamakata et al., 2020). These representations are attractive for fine-grained analysis of recipe texts but lack the expressivity to represent details such as adverbs or relations such as conditions and alternatives. Yet, this kind of information is important for correctly reconstructing the original meaning when generating instructions.

We explore generating recipe instructions at the action level from Abstract Meaning Representation (AMR) graphs. AMR is able to represent fine-grained and rich semantic relations, and its focus on predicate-argument structure makes it attractive for representing cooking instructions. Yet, AMR is a sentence-level representation that represents individual sentences in individual graphs.

This paper addresses the challenge of splitting sentence-level AMR graphs into the individual action-level AMRs. We present a splitting algorithm that considers the semantic relationships between actions in recipe instructions (§3)[1]. We evaluate our approach in a direct manual evaluation of the action-level representations as well as in an automatic and human evaluation of recipes generated from the created representation (§4). Findings

[1]Code and documentation is available at https://github.com/interactive-cookbook/recipe-generation

show that our algorithm accurately identifies action-level subgraphs in AMR recipe representations, suggesting its utility for AMR representations of procedural texts and for generating action-level instructions.

## 2 Related Work

At first glance, recipe texts may look quite simple. Yet recipe texts are semantically quite complex: subjects of actions are implicit, often as a result of being written in imperative mood; anaphoric expressions often refer to intermediate products that are outputs of actions and only partially corefer to input ingredients; and zero anaphora objects are frequent. To model recipe structure, most work on recipe text involves the identification and tagging of cooking actions, ingredients, intermediate substances and tools which get then used to create a structured representation (Mori et al., 2014a; Yamakata et al., 2020; Donatelli et al., 2021; Liu et al., 2022, *inter alia*). The most common representation approaches are graph and tree structures, which represent the flow and dependencies of actions and involved entities (Mori et al., 2014a; Jermsurawong and Habash, 2015; Kiddon et al., 2015; Yamakata et al., 2016, 2020; Donatelli et al., 2021).

The Abstract Meaning Representation (AMR) (Banarescu et al., 2013) framework represents the meaning of sentences with a focus on predicate-argument relationships, a key component of recipe structure. Figure 1a shows the AMR for the instruction *"Turn the dough out onto a surface dusted with flour and knead briefly until smooth."* on the left. As shown, AMRs are rooted, directed graphs in which nodes correspond to concepts and edges to the semantic relations between concepts. The framework makes use of a rich set of node and edge labels including frames from PropBank (Palmer et al., 2005), and within-sentence coreference is represented by re-entrancy.

Previous work on adapting AMR to the non-sentence level proposes approaches to create a multi-sentence AMR representation (O'Gorman et al., 2018; Naseem et al., 2022) or dialogue AMR graphs (Bai et al., 2021) but essentially keeps the sentence-level AMRs as part of the extended representations. AMR has also been used in the tasks of summarization (Liu et al., 2015; Lee et al., 2021) and text style transfer (Jangra et al., 2022).

## 3 Creating Action-Event AMRs

### 3.1 Actions and Action Events

Recipe texts should enable a cook to successfully prepare a dish by guiding them through the basic steps of the process. Yet, recipes rely on much commonsense knowledge for accurate interpretation, often combining several actions in one sentence or making only implicit reference to required actions. For example, *"**Turn** the dough **out** on a surface **dusted** with flour and **knead** until smooth."* conjoins the two steps of placing the dough on a surface and kneading it, mentioning the dusting of the surface only implicitly.

In previous work, the term *action* has been used in various ways. In some work, it refers to the action predicate together with its arguments (e.g. Kiddon et al., 2015; Liu et al., 2022). Often, only the action predicates themselves are referred to as an action (e.g. Mori et al., 2014b; Chang et al., 2018; Yamakata et al., 2020; Donatelli et al., 2021; Sakib et al., 2021). Trained taggers then identify corresponding spans of action predicate tokens.

To differentiate the two concepts, we use the term **action** to refer to an action predicate from here on and we introduce the concept of an **action event** to refer to an action predicate and all information belonging to it. In particular, we define an action event of a recipe as an individual action to be carried out by the cook together with all information (ingredients, time, result state, etc.) relevant to successfully complete the action. Importantly, not all actions in a sentence belong to different action events under this definition as we illustrate with examples from the action-tagged recipe corpus from Donatelli et al. (2021) shown in Table 1[2]. To distinguish actions and action events, we make use of the predicate-argument based structure of AMR.

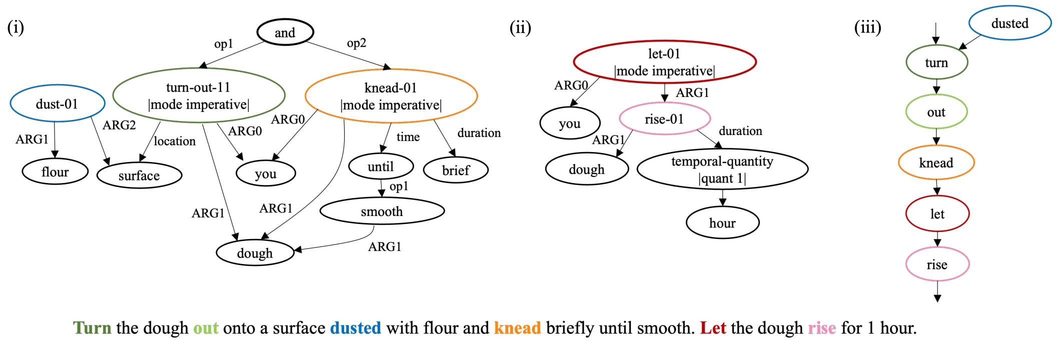
### 3.2 From Sentences to Action Events

AMR parsers typically predict one graph per sentence. Figure 1a presents two successive recipe instructions with the tagged actions shown in color; the corresponding **sentence-level AMRs** (**S-AMRs**) (i) and (ii); and a part of the action graph for the recipe (iii). The **action graph** consists of one node for each action and the edges represent their dependencies (Donatelli et al., 2021). Each of the two S-AMRs includes nodes corresponding to different actions. For both action and AMR graphs,

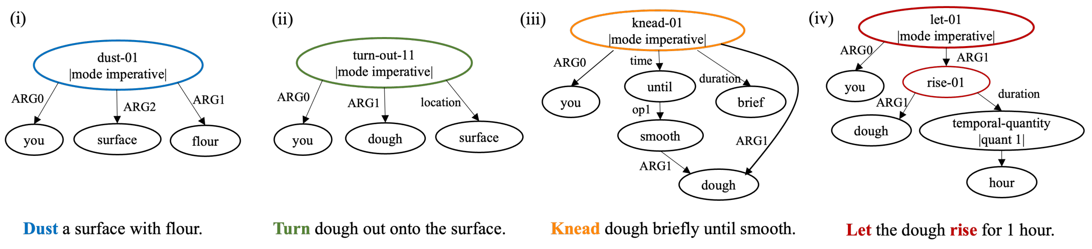
[2]Examples presented are shortened or slightly modified.

| | |
|---|---|
| (1) | **Place** **cooked** chicken on paper towel to **drain** the oil. |
| (2) | **Stir** in the chocolate chips by hand **using** a wooden spoon. |
| (3) | **Bake** for 30 minutes, or until a toothpick **comes out** clean. |
| (4) | Gradually **add** the water, while **mixing**. |
| (5) | **Let** the loafs **cool** for 10 minutes before **turning** onto a wire rack. |
| (6) | **Divide** the batter evenly among the mini loaf pans or **pour** into large loaf pan. |
| (7) | If it is still a little bit lumpy, you can **add** a touch of heavy cream, and **blend** again. |

Table 1: Examples of multi-action recipe instructions. Actions in the same color belong to the same action event.



(a) The S-AMRs for "Place the dough onto a surface dusted with flour and knead briefly until smooth." (i), "Let the dough rise for 1 hour." (ii) and a part of the action graph of the recipe to which the instructions belong (iii).


(i)
dust-01 |mode imperative|
ARG0
ARG2
ARG1
you
surface
flour
(ii)
turn-out-11 |mode imperative|
ARG0
ARG1
location
you
dough
surface
(iii)
knead-01 |mode imperative|
ARG0
time
duration
you
until
brief
op1
smooth
ARG1
ARG1
dough
(iv)
let-01 |mode imperative|
ARG0
ARG1
you
rise-01
ARG1
duration
dough
temporal-quantity |quant 1|
hour
Dust a surface with flour.
Turn dough out onto the surface.
Knead dough briefly until smooth.
Let the dough rise for 1 hour.

(b) The A-AMR graphs for the four action events "dust" (i), "turn out" (ii), "knead" (iii) and "let rise" (iv).

Figure 1: The different of graph representations we build upon in our work (1a) and the graphs we produce (1b).

we say that a node is **aligned** to the action (predicate) it corresponds to. Additionally, we say that an AMR graph is aligned to an action event if at least one node in the AMR is aligned to an action belonging to the action event.

Our goal is to split the S-AMRs into individual AMRs for action events (**A-AMRs**) such that each A-AMR includes exactly the actions from that event and all other nodes that belong to the action event, as shown in Figure 1b. Figure 1 illustrates several key aspects of this process. First, action aligned AMR nodes that belong to the same action event, such as `let-01` and `rise-01`, are always kept together. Second, each A-AMR contains only action nodes from a single action event. Different actions may share arguments as they operate on the same substances or tools, e.g. dough is the direct object of both "*turn out*" and "*knead*" and belongs to both action events. Our algorithm does not split an S-AMR into A-AMRs consisting of disjoint sets of nodes but selects the subgraph of the S-AMR that consists of nodes and edges belonging to the action event. Finally, the action graph allows us to properly order action events.

### 3.3 Datasets

As our main dataset, we use the ARA1.1 corpus[3] (Donatelli et al., 2021) which provides action graphs for 110 recipes spanning 10 dishes of the recipe corpus from Lin et al. (2020). We exclude three recipes and refer to the set of the remaining recipes as **ARA1**. We use an additional set of 110 recipes spanning 10 dishes as a secondary dataset (**ARA2**) to refine and validate our approach and use the tagger and parser from Donatelli et al. (2021)[4] to obtain the action tags and action graphs we use.

[3]https://github.com/interactive-cookbook/ara

[4]https://github.com/interactive-cookbook/tagger-parser

|  | Label node1 | LPath | Label node2 |
|---|---|---|---|
| 1) | *action* | ⟨ ARGX (opX)$^1$ ⟩ | *action* |
| 2) | *action* | ⟨ direction (opX)$^1$ ⟩ | *action* |
| 3) | *action* | ⟨ (*edge*)$^*$ *relation* (*edge*)$^*$ ⟩ | *action* |
|  | where *relation* is equal to purpose, manner, instrument, time or duration |  |  |
| 4) | *action* | ⟨ opX, opX-of ⟩ | *action* |
| 5) | *action* | ⟨ ARGX, ARGX-of ⟩ | *action* |

Table 2: Path patterns between two action-aligned AMR nodes that should be clustered together. *action* and *edge* can be any node or edge label, round brackets are used for optional labels on the **LPath**, ()$^1$ meaning zero or exactly one occurrence and ()$^*$ allowing any number of occurrences.

The full list of dishes and exclusion criteria can be found in Appendix A.

For our approach we rely on the availability of node-to-token alignments to determine to which action events each AMR is aligned. We obtain the S-AMRs for the recipes by parsing each sentence using the transition-based AMR parser of Drozdov et al. (2022)[5] (henceforth StructBART). The parser includes a neural aligner to predict node-to-token alignments needed for training and produces the alignments as a by-product during parsing.

## 3.4 Splitting Algorithm

Obtaining A-AMRs from S-AMRs consists of two main steps: (i) deciding which action-aligned AMR nodes correspond to the same action event and (ii) creating one A-AMR per action event from the S-AMR, i.e. extracting the appropriate subgraph. We focus on the overall process and the main decision rules in this section. The full set of clustering and splitting rules can be found in Appendix C.

Both steps of the process are based on the concepts of **labelled paths** (**LPath**) and **meeting nodes**. We define a path between two nodes $u$ and $v$ as a sequence of edges between two nodes where edges can be traversed in either direction and each node is visited only once. A labelled path is the sequence of the labels of edges of a path where edges that are traversed in reverse direction receive their reverse role label. For example, the LPath between dust-01 and turn-out-11 in the left AMR (i) in Figure 1a is ⟨ARG2, location-of⟩. We then define a meeting node as a node on a path at which two successive edges change their direction, i.e. where one edge is traversed in its original direction and the next edge in reverse direction or the other way round. On the path between dust-01 and and turn-out-11, there is one meeting node: surface.

The label of an edge between two action nodes represents the relation between them. LPaths allow us to capture relations between two action nodes that are further away, which we use to decide if two actions belong to the same action event. Meeting nodes are shared predecessor or successor nodes of two action nodes and intuitively correspond to nodes that belong to both action events such as shared arguments or conjunctions.

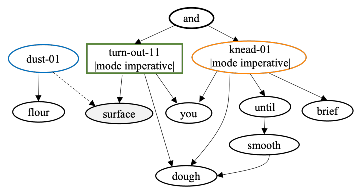
### 3.4.1 Clustering

Let $\mathbf{M_i}$ be the S-AMR for the i-th instruction in a recipe and let $\mathcal{A}$ be the set of all nodes of $\mathbf{M_i}$ aligned to different actions[6]. The clustering step groups all nodes $a \in \mathcal{A}$ into disjoint action clusters $\langle C_1, ...\rangle$ such that actions from the same action event are in the same cluster. It starts by creating all possible pairings of nodes from $\mathcal{A}$. Then for each pair $(a_i, a_j)$ all possible paths and LPaths between the nodes get computed and checked against a set of rules. Table 2 lists the main patterns used for the clustering rules: if the two action nodes $a_i, a_j$ and one of their LPaths match any of the patterns 1) - 3), $a_i$ and $a_j$ are clustered together. The patterns match the ways in which AMR represents the relations between actions of the same event. For example, 1) covers cases with discontinuous action spans and 3) can capture complex relations such as time specifications, even in nested structures.

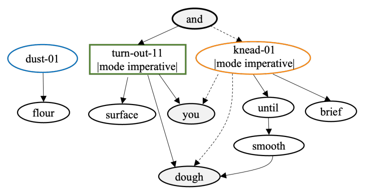
If pattern 4) or 5) matches, we check whether the meeting node is labelled or, slash or contrast-01 to make sure conjoined actions are not clustered together. Larger action clusters are built such that for each clustered pair $(a_i, a_j)$ $a_i$ and $a_j$ end up in the same final cluster $C_n$. If an S-AMR has or, slash, possible-01 or have-condition-91 as the root node, $\mathcal{A}$ is treated as a single action cluster.

[5]amr3.0-structured-bart-large-neur-al-sampling5-seed42 from https://github.com/IBM/transition-amr-parser

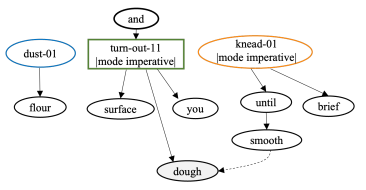
[6]If more than one AMR node is aligned to the same action we ignore the ones not labelled with a predicate frame.



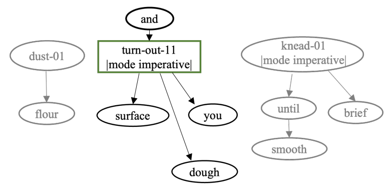
(a) Creating A-AMR for `turn-out` Step 1



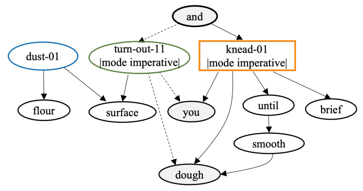
(b) Creating A-AMR for `turn-out` Step 2



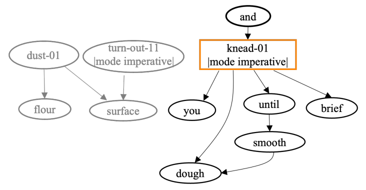
(c) Creating A-AMR for `turn-out` Step 3



(d) Creating A-AMR for `turn-out` Step 4



(e) Creating A-AMR for `knead` Step 1



(f) Creating A-AMR for `knead` Step 2

Figure 2: Step-by-step example of obtaining the A-AMRs from the S-AMR in Figure 1. Nodes from the current target cluster are shown as rectangles and edges that get removed as dotted edges. Creating the A-AMR for `dust` and the final postprocessing step are not shown. Edge labels are left out.

### 3.4.2 Splitting

If an S-AMR graph is aligned to more than one action cluster, the splitting algorithm is applied with the goal to obtain one A-AMR per cluster. Similar to the clustering approach, the splitting algorithm is based on the paths between action-aligned AMR nodes and meeting nodes.

The splitting algorithm always operates on one S-AMR and one target action cluster. The A-AMR gets created by iteratively removing nodes or edges until deriving one connected subgraph that contains all action nodes from the target cluster and no action nodes from any other clusters. Figure 3 presents the main structure of the algorithm: it starts by pairing each node from the target action cluster with all other action nodes, i.e. the pairs of all nodes that should not be connected anymore in the end. All paths from nodes of the target cluster to another cluster are considered for removing an edge or a node in order to separate the actions from each other. The shortest paths are considered first as they are usually the more meaningful paths that are captured by the removal conditions of the algorithm. The main rule checks for paths that consist of exactly one direction change, i.e. include one meeting node. If a path fulfills this condition then the edge "behind" the meeting node gets removed.

```
1. create a set $\mathcal{Q}$ of node pairs by pairing each action node from $C_j$ with each node from the other clusters
**repeat**
    2. create a sequence $\mathcal{P}$ of all paths for all pairs in $\mathcal{Q}$, ordered by length in ascending order
    3. **if** $\mathcal{P} = \emptyset$ then **break**
    4. **for** $p$ in $\mathcal{P}$ **do**
        (a) **if** CONDITION **do** REMOVAL and **continue** from 2.
    5. **if** the graph does not change anymore **then return** original graph
**end repeat**
6. select the connected subgraph that includes the nodes from the current cluster $C_j$ as the A-AMR for that action event
```

Figure 3: The main structure of the splitting algorithm given an S-AMR and a target action cluster $C_j$

Figure 2 illustrates the removal steps applied to derive the three A-AMRs (i-iii) from Figure 1b from the left S-AMR (i) in Figure 1a. The S-AMR includes three nodes aligned to an action node and the clustering results in three action clusters. Starting with the A-AMR for `turn-out`, one of the shortest paths connects `turn-out` and `dust` and consists of exactly one direction change at the shared child node `surface`. Therefore, the edge between (`surface`, `dust`) gets removed from the graph, as illustrated in Figure 2a. After removing an edge, the algorithm recomputes the set of all paths for the modified AMR.

There are still paths left in $\mathcal{P}$, so the splitting continues. Figure 2b and 2c summarize the next four iterations, in which the connecting paths between turn-out and knead get broken up. When no path between turn-out and any other action node is left then the connected component of the graph including that action gets selected as the A-AMR (see 2d).

The algorithm continues with the action cluster for knead. Removing the edges from the three shortest paths between knead and turn-out gets rid of all connecting paths resulting in the subgraph for knead (see 2f). Lastly, the A-AMR for dust gets created in the same way. A postprocessing step transforms the subgraphs into the final A-AMRs as shown in Figure 1b by removing redundant nodes and representing actions that originally were participles such as dust as imperatives.

An important characteristic of the algorithm is that it ensures that no nodes from the original S-AMR get lost except if removed by the rules themselves. If the splitting results in A-AMRs that do not cover all nodes from the original S-AMR, the S-AMR gets treated as non-separable. The same holds if any of the action clusters cannot be separated from all other clusters.

## 4 Evaluation

### 4.1 Manual Evaluation

We apply the splitting approach to the S-AMR graphs of the ARA1 and the ARA2 recipes. Table 3 provides an overview over the original datasets as well as the output of the splitting algorithm. The dataset of A-AMRs created by the splitting approach consists of 1396 AMR graphs for the ARA1 recipes out of which 584 A-AMRs are equivalent to the original S-AMR. For the ARA2 dataset the splitting results in a total of 1471 A-AMRs out of which 648 get not modified. Few S-AMRs cannot be separated into individual A-AMRs by our algorithm, proving its effectiveness.

To evaluate the splitting algorithm, we manually compare all original S-AMRs to the generated A-AMRs. In the ARA1 dataset we identified 64 A-AMRs that were incorrect relative to the source S-AMRs: either they were split incorrectly or not split although they should have been.[7] For 46 out of the 64 incorrect A-AMRs, the initial mistake already happens before the splitting process, i.e. in the action tagging step or during AMR parsing. In the ARA2 dataset, there are 68 incorrect A-AMRs

| | ARA1 | ARA2 |
|---|---|---|
| Recipes / action graphs | 107 | 110 |
| Action nodes | 1583 | 1771 |
| Sentences / S-AMRs | 941 | 1001 |
| Action clusters | 1391 | 1473 |
| A-AMRs | 1396 | 1471 |
| Non-separable S-AMRs | 14 | 13 |
| Incorrect A-AMRs | 64 | 68 |

Table 3: Overview of the ARA1 and ARA2 datasets (upper part) and the results from applying the splitting algorithm (lower part).

and for 58 of them the source mistake happens before the splitting step. We also identified cases for which the decision how to split the S-AMR is not straightforward. These cases will be discussed together with the limitations of the algorithm in Section 5.

### 4.2 NLG-based Evaluation

In addition to evaluating the splitting approach based on the output graphs themselves, we conduct a task-related evaluation. A potential use case for the fine-grained A-AMR graphs is the generation of recipe instructions at the action-event level in order to recombine them flexibly or present them incrementally to a user, e.g. to guide a user through the cooking process step by step in real-time. Therefore, we generate recipe instructions from the A-AMRs and evaluate them both automatically and manually with crowdsourced human evaluation.

To obtain gold instructions for the individual A-AMRs we use a rule-based heuristic. Another approach to obtain instructions corresponding to the A-AMRs would be to use an AMR-to-text model. However, as AMR parsers, AMR-to-text models are usually trained on the AMR3.0 corpus[8]. Therefore, the sentences generated by them for the A-AMRs might not resemble the style of recipe instructions (see §5). Splitting the instructions heuristically gives us a dataset on which we can fine-tune an AMR-to-text model for the recipe domain. Additionally, we can use the data to automatically evaluate and compare different models.

Our extraction heuristic is based on the node-to-token alignments produced by the parser and creates the gold instructions by selecting all tokens from the original instruction to which nodes in the specific A-AMR are aligned. Additionally, we use a set of rules based on POS tags to decide about the selection of unaligned tokens and to reorder

[7]We evaluate the "correctness" of the A-AMR given the S-AMR predicted by the StructBART parser.

[8]https://catalog.ldc.upenn.edu/LDC2020T02

| Model | BLEU | ROUGE-1 | ROUGE-2 | ROUGE-L | METEOR | BLEURT |
|---|---|---|---|---|---|---|
| amrlib | 44.00 | 74.9 | 45.96 | 68.88 | 72.38 | 38.28 |
| GART5-0 | 56.15 ±1.2 | 84.63 ±0.5 | 62.88 ±1.3 | 80.28 ±0.7 | 82.09 ±0.7 | 59.42 ±1.8 |
| GART5-1 | **57.84** ±1.3 | **85.75** ±0.4 | **65.98** ±0.9 | **81.91** ±0.7 | **83.47** ±0.6 | **62.64** ±1.5 |

Table 4: Results on the ARA1 test split averaged over 6 different seeds (± standard deviation).

|  | Grammar | Fluency | Verbosity | Structure | Success | Overall |
|---|---|---|---|---|---|---|
| Dependency | 2.88 | 2.58 | 3.26 | 3.33 | 3.40 | 2.76 |
| GART5-0 | 3.80 | 3.27 | 3.45 | 3.47 | 3.31 | 3.022 |
| GART5-1 | 3.62 | 3.02 | 3.15 | 3.20 | 3.01 | 2.74 |
| Original | 5.25 | 5.13 | 5.30 | 5.28 | 5.14 | 5.06 |

Table 5: Mean evaluation scores from the human evaluation per rating criterion for the recipes generated with context (GART5-1), without context (GART5-0) and by the dependency baseline and for the original recipe.

the selected tokens to improve the grammar (see Appendix A.1 for an example). Participle actions get stemmed to their imperative form. Overall, 812 and 823 of the A-AMRs of the ARA1 and ARA2 recipes get a new instruction out of which around 85% (ARA1) and 80% (ARA2) are grammatical.

#### 4.2.1 Generation Set-up

For the generation, we use the AMR-to-text model from the amrlib library[9] (amrlib model from here on), a pre-trained T5 model fine-tuned on the AMR3.0 corpus for AMR-to-text generation. We further fine-tune the amrlib model on the A-AMR dataset for the ARA1 recipes which we split into training (86), validation (11) and test (10) recipes.

Instead of passing a single linearized AMR graph we prepend the previous sentence as context information to the linearized AMR. For each recipe, we order the AMR-instruction pairs similarly to the instructions in the original recipe. A-AMRs obtained from the same S-AMR get ordered relative to each other based on the action graph such that e.g. *"Dust a surface with flour."* comes before *"Turn dough out onto the surface."*.

The amrlib model then gets fine-tuned to predict the sentence for the AMR-graph based on the AMR-graph and the context. Details about fine-tuning can be found in Appendix A. We call our generation model **GART5** (**G**enerating **A**ction-level **R**ecipes based on **T5**)[10].

#### 4.2.2 Automatic Evaluation

For the automatic evaluation, each A-AMR gets paired with the previous sentence from the original recipe as context. We then fine-tune our model on the graph-context pairs from the train recipes (GART5-1) and compare the results on the test recipes to two baselines: the texts generated by the original amrlib model[11] and instructions generated by a model fine-tuned on the recipe dataset without context (GART5-0). Additional ablation experiments can be found in Appendix A.2.

Table 4 presents the results of the automatic evaluation. Our GART5-0 model without context performs considerably better than the amrlib model on the A-AMR ARA1 test split across all metrics, achieving an improvement of 12 points in BLEU score and even 21 BLEURT points. Adding context in the fine-tuning step results in an additional - but smaller - improvement across all metrics.

#### 4.2.3 Crowd-sourcing Evaluation

In addition to the automatic evaluation, we conduct a human evaluation to get a more thorough and reliable assessment of the quality of the generated texts. 88 participants recruited via Prolific[12] judged various measures of coherence and acceptability for the generated instructions. Participants were paid £2.25 for their on average 15-minute participation.

We included each recipe from the test split in four versions. One version was generated with the GART5-0 and and one with the GART5-1 model, where the sentence generated at the previous time step was passed as context. As baseline recipes, we create action-level instructions from the original instructions by splitting them based on syntactic dependencies. Additionally, we include the original recipes as an upper bound. In the original condition, the instructions of each recipe were presented in

[9] https://github.com/bjascob/amrlib-models/releases/model_generate_t5wtense-v0_1_0

[10] Code for the training is available at https://github.com/interactive-cookbook/recipe-generation-model

[11] We remove the node-to-token alignments from the input to reproduce the format the amrlib model was trained on.

[12] https://www.prolific.co/

| **Dependency** | **GART5-0** | **GART5-1** |
|---|---|---|
| **Pour** over the flour mixture. | **Pour** in flour mixture. | **Pour** in flour mixture. |
| And very gently **stir**. | **Stir** very gently until about combined. | **Stir** very gently until about combined. |
| Until about **combin**. | | |
| **Melt** butter. | **Melt** butter. | **Melt** butter. |
| **Stir** in the butter. | **Stir** in butter. | **Stir** butter. |
| And **continue mixing** very gently until combined. | **Continue mixing** very gently until combined. | **Continue to mix** very gently until combined. |
| **Beat** egg whites until stiff. | **Beat** egg whites until stiff. | **Beat** in egg whites until stiff. |
| **Preheat** iron. | **Preheat** waffle iron. | **Preheat** waffle iron. |
| And slowly **fold** into batter. | Slowly **fold** in egg whites into batter. | Slowly **fold** in egg whites into the batter. |
| **Spoon** the batter into waffle iron in batches. | **Save** batter in batches on waffle iron. | **Scoop** the batter in batches onto waffle irons. |
| And **cook** according to its directions. | **Cook** batter according to directions. | **Cook** the batter according to its directions. |

| **Original** |
|---|
| **Pour** over the flour mixture and very gently **stir** until about **combined**. |
| **Stir** in the **melted** butter and **continue mixing** very gently until combined. |
| **Beat** egg whites until stiff and slowly **fold** into batter. |
| **Spoon** the batter into preheated waffle iron in batches and **cook** according to its directions. |

Table 6: An excerpt from a recipe for waffles in the four versions that were included in the crowd-sourcing evaluation.

their original order. In the other conditions, the order of the generated instructions was determined by traversing the corresponding action graph using a heuristic (see Appendix B).

Participants were presented two recipes per condition and they rated the textual quality of each recipe along six criteria on a six-point Likert Scale. Table 5 presents the results of the evaluation[13]. The original human written recipes were rated significantly better than the recipes from all other conditions for each rating criterion. Against our expectations and in contrast to the results of the automatic evaluation, we find that recipes generated with or without context were not rated significantly different with respect to their grammar, fluency and structure, but the recipes without context were rated significantly better with respect to their verbosity, success and overall quality. The grammar and fluency was rated worst in the dependency baseline.

## 5 Discussion

In this section we discuss the performance of our splitting algorithm and the results of the generation experiments in more detail.

**Splitting algorithm.** As described in §4.1, the splitting approach can successfully separate the S-AMRs of almost all instructions in the ARA1 and ARA2 recipes into A-AMRs. The iterative approach of removing edges at meeting nodes allows to split even deep and nested S-AMRs for long instructions successfully. For example, one of the instructions with the highest number of action events, *"**Remove** from oven and **let cool** on wire rack for about 10 minutes before **turning** bread **out** onto wire rack and **letting cool** completely before **slicing**, **toasting**, and **devouring**."* gets correctly separated into seven A-AMRs.

Many of the A-AMRs that are incorrectly split are based on a wrong S-AMR. We found that often the same tokens or specific types of tokens lead to parsing mistakes and that these tokens are mostly specific to the recipe domain (e.g. "grease", "Parmesan", "knead"). These observations are in line with the findings from Bai et al. (2021) that the main challenge for out-of-domain AMR parsing is the correct prediction of concepts. Additionally, when ignoring splitting mistakes resulting from parsing mistakes, almost all incorrect A-AMRs contain one of the following concepts: `mean-01`, `have-degree-91` and `have-quant-91`. These concepts are used to represent complex relations and they introduce path patterns into the AMR that are quite different and not covered by our algorithm.

Finally, during the manual evaluation of the A-AMRs we encountered a number graphs for which it is not straightforward to decide whether the specific splitting is adequate because of the specific semantic characteristics and especially temporal interactions of the actions. For example, *"**Bring** a*

[13]Statistical significance testing was performed using the software R (R Core Team, 2021) and the $lme4$ package (Bates et al., 2015). We used linear mixed effect models with condition as fixed effect, and by-subject and by-item intercepts and slopes as random effects, $p < 0.05$.

*pot of **salted** water to a boil."* gets split into *"**Salt** water."* and *"**Bring** a pot of water to a **boil**."*. However, none of the two potential orderings of the instructions is entirely adequate. On the one hand, the salt should be added before boiling the water. On the other hand, the water cannot be salted before it is filled into a pot but the "filling" action is only implicitly included in the original instruction.

**Generated texts.** In the automatic evaluation, fine-tuning on our recipe AMR dataset resulted in a considerable improvement compared to generating the instructions with the pretrained amrlib model. We found that the amrlib model struggles to produce the recipe specific writing style. For example, the amrlib model generates *"Stir for a commingling"* where the GART5 models generate *"Stir to combine"*.

In contrast to the automatic evaluation, we found that in the human evaluation the recipes generated with context were not judged significantly better. Table 6 shows an excerpt of a recipe for waffles in the four versions rated in human evaluation. Overall, the instructions generated by the two GART5 models are very similar. In our opinion, the most likely explanation for the different results is that the higher automatic evaluation scores are artifacts of the reference-based score computation and do not reflect real differences in quality.

The performance of the AMR parser also affected the quality of the generated texts as wrong concepts in the AMR lead to inadequate or nonsensical instructions. For example, representing "spoon" in the last instruction of the original version with `save-01` resulted in a wrong instruction generated by GART5-0.

**General discussion.** Our findings suggest that AMR representations are promising for representing and generating recipe instructions at the action level. The focus on predicate-argument structure makes them attractive for the representations of instructions as they center around actions and objects required to carry them out. Additionally, AMR graphs provide rich and fine-grained information about the semantic relations, the dependencies and also within-sentence coreference which makes it possible to identify the individual action events and to split even S-AMRs for long and nested instructions into their A-AMR components.

Furthermore, our approach produces again rich representations of the action events from which instructions for the individual action events can be generated. Heuristically splitting the textual instructions instead of the AMR representations would require a combination of different tools to predict all the relevant information such as dependencies and semantic roles. Additionally, splitting the instructions at the text level using our dependency baseline more often resulted in ungrammatical sentences as reflected by the significantly higher grammar and fluency ratings for the texts generated from the A-AMRs compared to the baseline.

## 6 Conclusion & Future Work

We have presented an approach to split sentence-level AMR representations for cooking recipe instructions into more fine-grained AMR representations of the individual action events. Our rule-based algorithm provides an automatic way to identify which cooking actions in a recipe instruction constitute separate action events to be carried out and to systematically breaking up the sentence-level AMRs into representations of the action events that provide more concise instructions. The predicate-argument oriented structure of AMR facilitates this process, and our approach achieves high performance on accurately breaking up the S-AMRs to more concise representations that can be used to generate instructions.

One bottleneck of our approach is the performance of the AMR parser in the domain of cooking recipes. Future work might investigate adapting AMR parsers to out-of-domain recipe vocabulary and processes. Regardless, representations of action events can support analysis and comparisons of actions in different cooking recipes as well as instruction generation in tasks that require more flexibility with respect to the exact order in which actions are instructed. As the presented approach makes use of the domain-independent structure of AMRs, we expect that it can generalize to other procedural texts, as well.

## Acknowledgments

We thank Iris Ferrazzo, Xiulin Yang and the members of the Interactive Cookbook project for fruitful discussions. We also thank the reviewers for their helpful comments. This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – KO 2916/3-1.

## References

Xuefeng Bai, Yulong Chen, Linfeng Song, and Yue Zhang. 2021. Semantic representation for dialogue modeling. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4430–4445, Online. Association for Computational Linguistics.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, pages 178–186.

Douglas Bates, Martin Mächler, Ben Bolker, and Steve Walker. 2015. Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48.

Minsuk Chang, Leonore V. Guillain, Hyeungshik Jung, Vivian M. Hare, Juho Kim, and Maneesh Agrawala. 2018. Recipescape: An interactive tool for analyzing cooking instructions at scale. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–12, New York, NY, USA. Association for Computing Machinery.

Lucia Donatelli, Theresa Schmidt, Debanjali Biswas, Arne Köhn, Fangzhou Zhai, and Alexander Koller. 2021. Aligning actions across recipe graphs. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6930–6942.

Andrew Drozdov, Jiawei Zhou, Radu Florian, Andrew McCallum, Tahira Naseem, Yoon Kim, and Ramón Astudillo. 2022. Inducing and using alignments for transition-based AMR parsing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1086–1098, Seattle, United States. Association for Computational Linguistics.

Anubhav Jangra, Preksha Nema, and Aravindan Raghuveer. 2022. T-star: truthful style transfer using amr graph as intermediate representation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, page 8805–8825, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Jermsak Jermsurawong and Nizar Habash. 2015. Predicting the structure of cooking recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 781–786, Lisbon, Portugal. Association for Computational Linguistics.

Chloé Kiddon, Ganesa Thandavam Ponnuraj, Luke Zettlemoyer, and Yejin Choi. 2015. Mise en place: Unsupervised interpretation of instructional recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 982–992, Lisbon, Portugal. Association for Computational Linguistics.

Fei-Tzin Lee, Chris Kedzie, Nakul Verma, and Kathleen McKeown. 2021. An analysis of document graph construction methods for amr summarization. *arXiv preprint arXiv:2111.13993*.

Angela Lin, Sudha Rao, Asli Celikyilmaz, Elnaz Nouri, Chris Brockett, Debadeepta Dey, and Bill Dolan. 2020. A recipe for creating multimodal aligned datasets for sequential tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4871–4884, Online. Association for Computational Linguistics.

Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A. Smith. 2015. Toward abstractive summarization using semantic representations. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1077–1086, Denver, Colorado. Association for Computational Linguistics.

Xiao Liu, Yansong Feng, Jizhi Tang, Chengang Hu, and Dongyan Zhao. 2022. Counterfactual recipe generation: Exploring compositional generalization in a realistic scenario. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7354–7370, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Manuel Mager, Ramón Fernandez Astudillo, Tahira Naseem, Md Arafat Sultan, Young-Suk Lee, Radu Florian, and Salim Roukos. 2020. GPT-too: A language-model-first approach for AMR-to-text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1846–1852, Online. Association for Computational Linguistics.

Shinsuke Mori, Hirokuni Maeta, Tetsuro Sasada, Koichiro Yoshino, Atsushi Hashimoto, Takuya Funatomi, and Yoko Yamakata. 2014a. FlowGraph2Text: Automatic sentence skeleton compilation for procedural text generation. In *Proceedings of the 8th International Natural Language Generation Conference (INLG)*, pages 118–122, Philadelphia, Pennsylvania, U.S.A. Association for Computational Linguistics.

Shinsuke Mori, Hirokuni Maeta, Yoko Yamakata, and Tetsuro Sasada. 2014b. Flow graph corpus from recipe texts. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 2370–2377, Reykjavik, Iceland. European Language Resources Association (ELRA).

Tahira Naseem, Austin Blodgett, Sadhana Kumaravel, Tim O'Gorman, Young-Suk Lee, Jeffrey Flanigan, Ramón Astudillo, Radu Florian, Salim Roukos, and Nathan Schneider. 2022. DocAMR: Multi-sentence AMR representation and evaluation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3496–3505, Seattle, United States. Association for Computational Linguistics.

Tim O'Gorman, Michael Regan, Kira Griffitt, Ulf Hermjakob, Kevin Knight, and Martha Palmer. 2018. AMR beyond the sentence: the multi-sentence AMR corpus. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3693–3702, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.

R Core Team. 2021. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Leonardo F. R. Ribeiro, Martin Schmitt, Hinrich Schütze, and Iryna Gurevych. 2021. Investigating pretrained language models for graph-to-text generation. In *Proceedings of the 3rd Workshop on Natural Language Processing for Conversational AI*, pages 211–227, Online. Association for Computational Linguistics.

Md Sadman Sakib, Hailey Baez, David Paulius, and Yu Sun. 2021. Evaluating recipes generated from functional object-oriented network. *CoRR*, abs/2106.00728.

Yoko Yamakata, Shinji Imahori, Hirokuni Maeta, and Shinsuke Mori. 2016. A method for extracting major workflow composed of ingredients, tools, and actions from cooking procedural text. In *2016 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 1–6.

Yoko Yamakata, Shinsuke Mori, and John Carroll. 2020. English recipe flow graph corpus. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 5187–5194, Marseille, France. European Language Resources Association.

## A Experiments

### A.1 Model Training and Evaluation

**Dataset.** We restrict the recipes for our work to those recipes of the ARA1 corpus that describe the preparation of only one dish in a continuous text. Three of the 110 recipes do not meet this criterion and get excluded. The ten recipes for the test split were chosen manually based on the criteria that the original recipe text should not be shorter than 6 sentences and not include a lot of additional information or noise (e.g. nutrition lists). In order to avoid selecting recipes that are particularly easy for our approach, the recipes were selected by a student who was not familiar with the performance of the different parts of the pipeline on different kind of instructions and linguistic constructions. The test split consists of one recipe for each of the ten ARA1 dishes and comprises 151 A-AMR - sentence pairs in total. The remaining 97 recipes were randomly split into training and validation data.

| **ARA1** |
|---|
| Baked Ziti, Blueberry Banana Bread, Cauliflower Mash, Chewy Chocolate Chip Cookies, Garam Masala, Homemade Pizza Dough, Orange Chicken, Pumpkin Chocolate Chip Bread, Slow Cooker Chicken Tortilla Soup, Waffles |
| **ARA2** |
| Bananas Foster, Chocolate Glaze, Cobb Salad, English Muffin Bread, Homemade Graham Crackers, How to Roast Garlic, Lavender Lemonade, Peanut Butter Bars, Sausage Grave, Southern Sweet Tea |

Table 7: List of the Dishes from ARA1 and ARA2.

**Gold instructions.** Our extraction heuristic makes use of the node-to-token alignments produced by the AMR parser. Figure 4 shows the A-AMRs resulting from splitting the S-AMR for *"Top with shredded mozzarella cheese"* in PENMAN notation. We obtain the two corresponding gold instructions *"Shred mozzarella cheese"* and *"Top with mozzarella cheese"* by selecting all tokens from the original instruction to which nodes in the specific A-AMR are aligned. Tokens that have alignments to nodes in more than one A-AMR from the same S-AMR are included in the gold instruction for each of the A-AMRs (e.g. "mozzarella" and "cheese".) We use a set of rules based

```
(d / top~e.135
   :mode imperative~e.135
   :ARG0 (y / you~e.135)
   :ARG2 (c / cheese~e.139
            :mod (m / mozzarella~e.138)))
```

```
(s / shred-01~e.137
   :mode imperative~e.137
   :ARG1 (c / cheese~e.139
            :mod (m / mozzarella~e.138))
   :ARG0 (y / you~e.137))
```

| **Top** | with | **shredded** | mozzarella | cheese | . |
|---|---|---|---|---|---|
| 135 | 136 | 137 | 138 | 139 | 140 |

Figure 4: The A-AMRs obtained from the S-AMR for "Top with shredded mozzarella cheese." with the node-to-token alignments. The IDs of tokens that only have alignments to nodes in the A-AMR for "top" or for "shred" are marked in orange and blue respectively, and the IDs of tokens with alignments in both A-AMRs are green.

on patterns of the POS tags of successive tokens to decide about the selection of tokens that do not have an aligned node in any of the A-AMRs such as prepositions and determiners. Participle actions such as "shredded" get stemmed to ensure that the extraction heuristic creates grammatical imperative instructions. Additionally, the tokens selected for a gold instruction get partially reordered such that the action predicate is at the correct position in the new sentence if possible.

**Training details.** For training we build on the training scripts from the amrlib library[14] and adapt the required input format to our training set-up. All models are trained using the Adam optimizer and a linear learning rate scheduler with warm-up with $1e-4$ as the initial learning rate. The dropout rate is set to 0.1 for all reported experiments. The training and validation batch size is set to 24. We train all models using early stopping based on the train loss with a patience of 15 and a threshold of 0.00005 and select the final model based on the best BLEU score on the validation set.

Following previous work on using transformer LMs for AMR-to-text generation, we use a graph linearization based on the PENMAN format of the AMRs (Mager et al., 2020; Ribeiro et al., 2021, *inter alia*). We create the input to the model by concatenating the context sentence and the AMR in PENMAN format including the node-to-token alignments and introduce a special token to separate the context and the graph (see Table 8).

**Generation.** We set the token limitation for each generated sequence to 1024 and let the model output the best sequence using a beam size of 1.

**Automatic evaluation.** We compute BLEU, Rouge-1 (R-1), Rouge-2 (R-2), Rouge-L, Meteor (M) and Bleurt (BLRT) scores. For the computation of all automatic metrics we use the Hugging-face Evaluate Metric package[15] which provides wrappers around the original metric implementations or the implementations from the SacreBLEU tool for comparable evaluation scores. We leave all parameters at their default values. For BLEU, we compute case-insensitive BLEU-4 at the corpus level. For BLEURT, we use the pre-trained bleurt-large-512 checkpoint and average over all predicted sentence-level scores to obtain a final BLEURT score.

[14]https://github.com/bjascob/amrlib

[15]https://huggingface.co/evaluate-metric

## A.2 Ablation Experiments

**Unseen dishes.** The test recipes are highly related to the training recipes as they are for the same dishes. In order to assess the performance of our models on new, unseen dishes, we evaluate the same GART5-0 and GART5-1 models also on the complete ARA2 recipes. The amrlib model performs worse than the GART5-0 model on all metrics on the ARA2 recipes, showing the general benefit of fine-tuning the generation model on a similar dataset from the same domain. However, the improvement is around 50% smaller than on the ARA1 recipes (see Table 9).

**Effect of AMR type and alignments.** We conducted some additional experiments to assess if and how the differences between the original S-AMRs and the split A-AMRs affect the performances of the models and to what extent the inclusion of the node-to-token alignments has an effect. Table 10 presents the results of training the GART5-0 and GART5-1 models on the A-AMR and the S-AMR datasets and testing on the same or the other dataset (approximately 40% of the amr-sentence pairs from the A-AMR dataset are also included in the S-AMR dataset). Overall, the results indicate that training and testing on the same kind of dataset yields the best results.

**GART5-0**: <GRAPH> (t / top∼e.135 :ARG2 (c / cheese∼e.139 :mod (m / mozzarella∼e.138)))
**GART5-1**: Shred mozzarella cheese <GRAPH> (t / top∼e.135 :ARG2 (c / cheese∼e.139 :mod (m / mozzarella∼e.138 )))
**Output**: Top with the mozzarella cheese.

Table 8: Example of the input to the GART5 generation model without context and with context and of a generated output sentence.

| Model | BLEU | R-1 | R-2 | R-L | M | BLRT |
|---|---|---|---|---|---|---|
| amrlib | 41.69 | 75.56 | 45.12 | 69.75 | 73.26 | 48.06 |
| GART5-0 | 47.78 ±0.4 | 80.71 ±0.2 | 53.39 ±0.5 | 76.30 ±0.3 | 77.86 ±0.3 | 58.33 ±0.7 |
| GART5-1 | **51.08** ±0.5 | **82.23** ±0.3 | **57.57** ±0.6 | **77.95** ±0.3 | **79.63** ±0.4 | **59.2** ±0.6 |

Table 9: Results on the full ARA2 dataset averaged over 6 different seeds (± standard deviation).

Regarding the effect of keeping the node-to-token alignments in the linearization we did not have any specific hypothesis. On the one hand, the amrlib model did not include alignments which could lead to a lower performance. On the other hand, the alignments indicate the original relative order of the words corresponding to the nodes and they might help the model to generate a well-ordered sentence. Table 11 presents the results from training and testing with and without the alignments in the graph linearization. The models trained and tested on the PENMAN linearization including the alignments perform best or second best across all metrics.

## B Human Evaluation Set-up

**Ordering heuristic.** A correctly ordered sequence of the nodes $a \in \mathcal{N}_\mathbf{A}$ of an action graph needs to be a topological ordering of the action graph. However, not all potential orderings are good for structuring the steps in a recipe. For the generated recipes used in the human evaluation we defined a heuristic for ordering the actions that is based on the intuition that it is more convenient to keep working one subprocess for several steps instead of switching back and forth between different subprocesses. The traversal produces the ordered sequences $\mathcal{T}$ of action nodes in the following way:

1. Consider the set $\mathcal{B}$ of all nodes $a$ without a parent node
2. Start the traversal with that node $a_i \in \mathcal{B}$ for which $\mathbf{Path}(a_i, end)$ is longest
3. Traverse the graph and add each visited node to $\mathcal{T}$ until reaching a node $a_j$ that has parent nodes that are not yet in $\mathcal{T}$
4. Consider all nodes $a_k \in \mathcal{B}$ and $a_k \notin \mathcal{T}$ for which there is a $\mathbf{Path}(a_k, a_j)$ and select the node for which the path is longest to continue the traversal. If there are two candidate nodes chose the one which occurs earlier in the recipe text

**Dependency baseline.** The instructions for the baseline recipes are obtained based on the syntactic dependency tree of each instruction. The dependency splitting approach creates one instruction for each individual action predicate because the clustering approach to identify action events is based on semantic relations that are not available from the plain text. For each action, the baseline instruction is generated by selecting all tokens that can be reached by traversing the dependency tree starting from the action predicate without passing another action. We then use the same approach as for generating the gold instructions for re-ordering tokens and stemming participle actions.

**Evaluation.** For the generation of the recipe instructions presented in the human evaluation we used the specific checkpoints for which the automatic evaluation results are shown in Table 12. Table 13 presents the statements that were presented to the participants in the human evaluation study. Each participant saw one recipe at a time followed by the six statements. They were asked to rate for each of them to what extent they agree with the statement on a scale from 1 (disagree completely) to 6 (agree completely). In order to ensure that participants did pay attention to the recipe texts we included two filler recipes: one including multiple grammatical mistakes and one with randomly ordered instructions. The data from participants who rated the first one with a six or the latter one with 5 or higher was not included in the evaluation resulting in data from 88 participants.

## C Clustering and Splitting

Table 5 lists rules that are used during the pairwise action clustering to decide whether two action nodes belong to the same event as well as the rules

| Model | Train | Test | BLEU | R-1 | R-2 | R-L | M | BLRT |
|---|---|---|---|---|---|---|---|---|
| GART5-0 | A-AMR | A-AMR | 54.10 | 84.32 | 61.85 | **79.98** | 81.17 | 58.23 |
| GART5-0 | S-AMR | S-AMR | **54.86** | **84.64** | **64.58** | 79.09 | **82.58** | **59.81** |
| GART5-0 | A-AMR | S-AMR | 52.08 | 83.19 | 61.09 | 78.76 | 79.92 | 53.39 |
| GART5-0 | S-AMR | A-AMR | 53.99 | 83.8 | 59.92 | 78.79 | 81.34 | 56.57 |
| GART5-1 | A-AMR | A-AMR | **59.26** | **86.34** | **67.78** | **83.19** | **84.25** | **65.14** |
| GART5-1 | S-AMR | S-AMR | 58.15 | 85.34 | 67.52 | 80.03 | 82.95 | 59.86 |
| GART5-1 | A-AMR | S-AMR | 57.01 | 85.19 | 66.87 | 81.62 | 82.85 | 62.40 |
| GART5-1 | S-AMR | A-AMR | 55.82 | 84.46 | 63.33 | 79.40 | 81.53 | 57.51 |

Table 10: Comparisons of the performance of the models when trained and tested on the A-AMR or S-AMR datasets.

| Model | Train | Test | BLEU | R-1 | R-2 | R-L | M | BLRT |
|---|---|---|---|---|---|---|---|---|
| GART5-0 | wA | wA | 54.10 | **84.32** | 61.85 | **79.98** | 81.17 | **58.23** |
| GART5-0 | nA | nA | **54.66** | 83.57 | **61.87** | 79.52 | 80.73 | 56.16 |
| GART5-0 | wA | nA | 52.08 | 83.19 | 61.09 | 78.76 | 79.92 | 53.39 |
| GART5-0 | nA | wA | 53.59 | 82.73 | 60.67 | 78.29 | **81.23** | 56.88 |
| GART5-1 | wA | wA | **59.26** | **86.34** | **67.78** | **83.19** | **84.25** | **65.14** |
| GART5-1 | nA | nA | 58.29 | 85.57 | 65.92 | 82.26 | 83.17 | 61.04 |
| GART5-1 | wA | nA | 58.17 | 85.12 | 65.32 | 81.84 | 82.82 | 61.92 |
| GART5-1 | nA | wA | 57.57 | 85.08 | 65.54 | 81.91 | 83.97 | 63.61 |

Table 11: Comparison of performances for different graph linearizations: with node-to-token alignments (wA) and without (nA).

for deciding already based on the root node that an S-AMR will not get separated. The full set of path patterns used by the rules are presented in Table 14.

In §3 we presented the main parts of the splitting algorithm. Table 6 presents the full algorithm with all rules and special cases. The following notation is used to describe the splitting conditions: When describing a path of actions between two nodes that includes and edge $e_k$ we use the notation $e_k^{\rightarrow}$ and $e_k^{\leftarrow}$ to differentiate between edges that are traversed in their original direction ($e_k^{\rightarrow}$) and edges that are traversed in the reverse direction ($e_k^{\leftarrow}$). Therefore, if a path includes $\langle ..., e_k^{\rightarrow}, e_l^{\rightarrow}, ... \rangle$ with $e_k = (u, v) and e_l = (v, w)$ (i.e. the original edge in the graph is $(w, v)$) then $v$ is a meeting node.

| Model | Test context | BLEU | R-1 | R-2 | R-L | M | BLRT |
|---|---|---|---|---|---|---|---|
| GART5-0 | 0 | 54.10 | 84.32 | 61.85 | 79.98 | 81.17 | 58.23 |
| GART5-1 | 1 | **59.26** | **86.34** | **67.78** | **83.19** | **84.25** | **65.14** |

Table 12: Results of the specific checkpoints used to generate the texts for the human evaluation on the ARA1 test split.

| Criterion | Statement |
|---|---|
| Grammar | The recipe text is grammatically correct. |
| Fluency | The recipe text reads smoothly. |
| Verbosity | The recipe explains the steps concisely and does not repeat information unnecessarily. |
| Structure | The recipe explains the steps in a helpful order. |
| Success | In combination with a list of the required ingredients, the recipe would enable me to successfully prepare the dish. |
| Overall | Overall, the recipe is well written. |

Table 13: The statements used in the human evaluation to assess the quality of the recipes along different criteria.

**Root-based rules:**

Let $\mathbf{M_i}$ be an S-AMR with root node $r_{\mathbf{M_i}}$. Do not split $\mathbf{M_i}$ if one of the following holds:

- the label of the root is `or`, `slash`, `possible-01` or `have-condition-91`
- the root node has an outgoing edge $(r_{\mathbf{Mi}}, u)$ to any node $u$ with the label `condition`

**Action-pair based rules:**

Let $\mathbf{M_i}$ be a S-AMR and $a_1, a_2$ two action nodes of $\mathbf{M_i}$ aligned to different actions:

**1.** Pair $a_1$ and $a_2$ into one action cluster if there exists a path $\mathbf{Path}(a_1, a_2)$ which does not include any direction changes and if for the corresponding labelled path $\mathbf{LPath}$ one of the following conditions holds:

- The $\mathbf{LPath}$ corresponds to one of the path patterns from Pattern Set1 in Table 14, with $a_1$ and $a_2$ corresponding to **Node1** and **Node2**
- The $\mathbf{LPath}$ corresponds to one of the path patterns from Pattern Set2 in Table 14, with $a_1$ and $a_2$ corresponding to **Node1** and **Node2** and one of the following conditions holds
  - the path $\mathbf{Path}(a_1, a_2)$ between the two action nodes does not contain a node $v$ that is labelled `before` or `after`
  - the path $\mathbf{Path}(a_1, a_2)$ between the two action nodes contains a node $v$ that is labelled `before` or `after` and $v$ has more than one child node.

**2.** Pair $a_1$ and $a_2$ into one action cluster if there exists a path $\mathbf{Path}(a_1, a_2)$ with exactly one direction change, i.e. with one meeting node $v$, and if one of the following conditions holds for the corresponding labelled path $\mathbf{LPath}$ and the meeting node:

- The $\mathbf{LPath}$ corresponds to the first pattern of Pattern Set3 in Table 14 and $v$ is labelled `or` or `slash`
- The $\mathbf{LPath}$ corresponds to the second pattern of Pattern Set3 in Table 14 and $v$ is labelled `contrast-01`

Figure 5: Rules for the pairwise clustering of action nodes of an S-AMR into action-event clusters.

| | Label node1 | LPath | Label node2 |
|---|---|---|---|
| Pattern Set1 | *action* | ⟨ ARGX (opX)$^1$ ⟩ | *action* |
| | *action* | ⟨ direction (opX)$^1$ ⟩ | *action* |
| | *action* | ⟨*edge*⟩ | off\|up\|down\|out\|in |
| | stir-01 | ⟨*edge*⟩ | fry-01 |
| Pattern Set2 | *action* | ⟨ (*edge*)$^*$ *relation* (*edge*)$^*$ ⟩ | *action* |
| | where *relation* is equal to purpose, manner, instrument, time or duration | | |
| Pattern Set3 | *action* | ⟨ opX, opX-of ⟩ | *action* |
| | *action* | ⟨ ARGX, ARGX-of ⟩ | *action* |

Table 14: Path patterns between two action-aligned AMR nodes that should be clustered together. *action* and *edge* can be any node or edge label, round brackets are used for optional labels on the $\mathbf{LPath}$, ()$^1$ meaning zero or exactly one occurrence and ()$^*$ allowing any number of occurrences.

```
Input: a copy $\mathbf{N_i}$ of an S-AMR graph $\mathbf{M_i}$, the target action cluster $C_j$, and the set of all other action clusters $C_k, k \neq j$
Output: an A-AMR graph for $C_j$ if successful, else the original S-AMR
1. create the set $\mathcal{Q}$ of all pairs $\{(a_1, a_2)|a_1 \in C_j$ and $a_2 \in \bigcup_{k \neq j} C_k\}$, i.e. all pairs of action AMR nodes that need to get separated from each other
repeat
  2. compute all paths $p = \mathbf{Path}(a_1, a_2)$ for all pairs $(a_1, a_2) \in \mathcal{Q}$ in the graph $\mathbf{N_i}$ and create a sequence $\mathcal{P}$ of all paths ordered by length in ascending order
  3. if $\mathcal{P} = \emptyset$ then break because then all nodes from $C_j$ are successfully separated from all other action clusters
  4. for $p$ in $\mathcal{P}$ do
    4.1 if $p$ does not include any node $u$ labelled before or after
      4.1.1 if $p$ has exactly one direction change ($\rightarrow$ to $\leftarrow$ or $\leftarrow$ to $\rightarrow$), and ($p = \langle ..., e_k^{\leftarrow}, e_l^{\rightarrow}, ...\rangle$ or $p = \langle ..., e_k^{\rightarrow}, e_l^{\leftarrow}, ...\rangle$) with $e_k = (v, w)$ and $e_l = (w, x)$, i.e. $w$ is the meeting node, then remove $e_l$ from $\mathbf{N_i}$ and continue from step 2.
    4.2 else $p$ includes a node $u$ labelled before or after
      4.2.1 if $p$ has no direction changes then remove $u$ from $\mathbf{N_i}$ and continue from step 2.
      4.2.2 else if $p$ has exactly one direction change ($\leftarrow$ to $\rightarrow$), and $p = \langle ..., e_k^{\leftarrow}, e_l^{\rightarrow}, ...\rangle$ with $e_k = (v, w)$ and $e_l = (w, x)$, i.e. $w$ is the meeting node, and $\mathcal{L}_{\mathbf{Mi}}(w) =$ and then remove $u$ from $\mathbf{N_i}$ and continue from step 2.
      4.2.3 else if $p$ has exactly one direction change ($\rightarrow$ to $\leftarrow$ or $\leftarrow$ to $\rightarrow$), and ($p = \langle ..., e_k^{\leftarrow}, e_l^{\rightarrow}, ...\rangle$ or $p = \langle ..., e_k^{\rightarrow}, e_l^{\leftarrow}, ...\rangle$) with $e_k = (v, w)$ and $e_l = (w, x)$, i.e. $w$ is the meeting node, then remove $e_l$ from $\mathbf{N_i}$ and continue from step 2.
  5. for $p$ in $\mathcal{P}$ do (fallback case if $\mathbf{N_i}$ did not change during step 4.)
    5.1 if $p$ has more than one direction change, and ($p = \langle ..., e_k^{\leftarrow}, e_l^{\rightarrow}, ..., e_o^{\rightarrow}, e_p^{\leftarrow}, ...\rangle$ or $p = \langle ..., e_k^{\leftarrow}, e_l^{\rightarrow}, ..., e_o^{\leftarrow}, e_p^{\rightarrow}, ...\rangle$) with $e_k = (v, w)$, $e_l = (w, x)$ and $w$ being the first meeting node and $\mathcal{L}_{\mathbf{Ni}}(w) =$ and and with $e_o = (y, z)$, $e_p = (z, z_2)$ and $z$ being the last meeting node then remove $e_p$ from $\mathbf{N_i}$ and continue from step 2
    5.2 else if $p$ has more than one direction change, and ($p = \langle ..., e_k^{\leftarrow}, e_l^{\rightarrow}, ...\rangle$ or $p = \langle ..., e_k^{\rightarrow}, e_l^{\leftarrow}, ...\rangle$) with $e_k = (v, w)$, $e_l = (w, x)$ and $w$ being the first meeting node then remove $e_l$ from $\mathbf{N_i}$ and continue from step 2
  6. if $\mathbf{N_i}$ did not change during step 5. then return original graph $\mathbf{M_i}$
end repeat
7. select the connected subgraph that includes all nodes from the target cluster $C_j$ as the new action-event-level AMR, apply the postprocessing and return the graph
```

Figure 6: The full splitting algorithm.