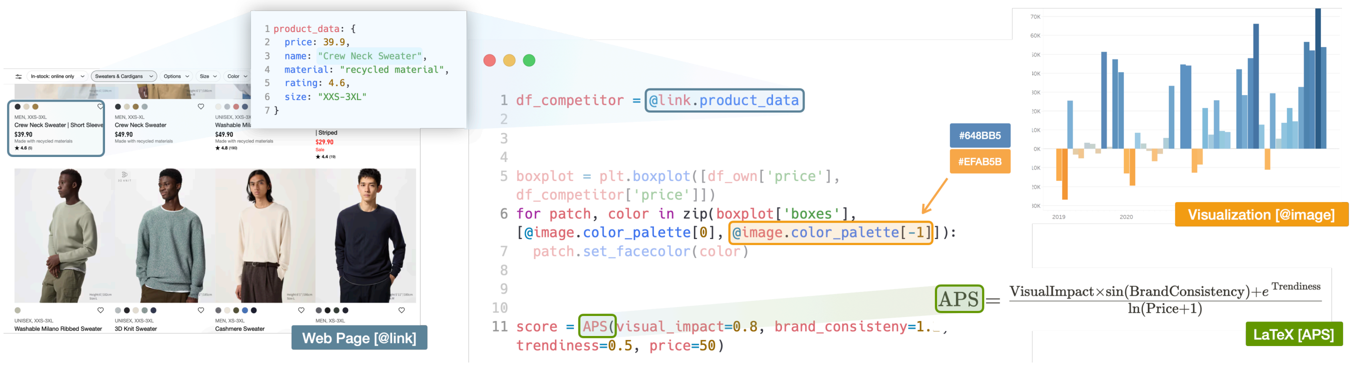
Something In Between Formal Spec and Informal Representation

Ryan Yen
MIT CSAIL
ryanyen2@mit.edu

Josh Pollock
MIT CSAIL
jopo@mit.edu

Caroline Berger
Aarhus University
caroline.berger@cs.au.dk

Arvind Satyanarayan
MIT CSAIL
arvindsatya@mit.edu



```
product_data: {
  price: 39.9,
  name: "Crew Neck Sweater",
  material: "recycled material",
  rating: 4.6,
  size: "XXS-3XL"
}
```

```
df_competitor = @link.product_data



boxplot = plt.boxplot([df_own['price'],
df_competitor['price']])
for patch, color in zip(boxplot['boxes'],
[@image.color_palette[0], @image.color_palette[-1]]):
  patch.set_facecolor(color)




score = APS(visual_impact=0.8, brand_consisteny=1._,
trendiness=0.5, price=50)
```

$$\mathrm{APS} = \frac{\mathrm{VisualImpact} \times \sin(\mathrm{BrandConsistency}) + e^{\mathrm{Trendiness}}}{\ln(\mathrm{Price}+1)}$$

Figure 1: A scenario illustrating a semi-formal paradigm, where programmers code with web page, image, and *latex* formula.

Programming is not merely the act of writing rigid code syntax. In practice, programmers often engage with a range of informal representations, like sketching architectures on whiteboards, drawing flowcharts to plan logic, or collecting visual examples from the web. These artifacts are not just peripheral aids; they are central to how programmers think, communicate, and iterate. Programmers constantly move between these informal representations and formal code, using each to inform and refine the other [1].

However, existing programming environments often attempt to incorporate these informal artifacts by strictly formalizing them into executable code too early or by superficially overlaying formal specifications onto informal content. Systems like Sketch-n-Sketch employ program synthesis to translate user manipulations directly into formal code [2], whereas tools such as Inkbase overlay predefined attributes onto informal sketches incrementally [3]. While these methods effectively bridge informal and formal representations within certain domains, they suffer from limited generalizability. Premature formalization frequently imposes rigid structures that restrict adaptability, impeding scalability and flexibility [5].

In response, we propose an alternative approach, *semi-formal paradigm*, which operates dynamically between formal and informal spaces. This paradigm leverages foundation model to reason about evolving user intentions, dynamically extracting attributes from informal representations and incrementally structuring them based on context. Central principles include gradually enriching informal artifacts into structured data, relaxing strict type constraints to accommodate fuzzy types [4], and enabling just-in-time resolution of ambiguities through context-sensitive AI reasoning.

Consider a concrete scenario of brand analysis (see Figure 1). Initially, the user simply pastes a web URL and references it informally as `@link.product_data` , allowing AI-driven scraping to dynamically extract structured details from the web page, such as product prices, clarifying ambiguities as needed. Next, the user informally extracts aesthetic attributes, like color palettes from an uploaded image. They used references such as `@image.color_palette[-1]` , integrating the extracted color value into their own visualizations. Finally, informal *LaTeX* annotations, such as the Aesthetic-Price Score (APS) formula defined in a document, transformed into executable computations such as `APS()` , enabling users to reuse and parameterize the function.

Substantial research remains necessary to fully realize this semi-formal paradigm. Exploring how users navigate between formal and informal representations, developing efficient mechanisms for dynamic attribute extraction, and refining AI-driven context reasoning remain open challenges. Our ongoing work aims to address these areas, ultimately creating programming tools that support human dynamic cognition: fluid, iterative, and continuously evolving. This vision aligns closely with the broader goals of the CHI'25 Tools for Thought workshop. By fostering systems that dynamically bridge informal thought and formal code syntax, our semi-formal paradigm directly engages core questions around protecting and augmenting critical thinking and enabling new sensemaking strategies towards program.

For more details and video demos, please refer to this link.

, ,
.

REFERENCES

[1] Mauro Cherubini, Gina Venolia, Rob DeLine, and Andrew J. Ko. 2007. Let's go to the whiteboard: how and why software developers use drawings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 557–566. https://doi.org/10.1145/1240624.1240714

[2] Brian Hempel, Justin Lubin, and Ravi Chugh. 2019. Sketch-n-Sketch: Output-directed programming for SVG. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 281–292. https://doi.org/10.1145/3332165.3347925

[3] James Lindenbaum, Szymon Kaliski, and Joshua Horowitz. 2022. Inkbase: Programmable Ink. https://www.inkandswitch.com/inkbase

[4] Cyrus Omar, Ian Voysey, Michael Hilton, Jonathan Aldrich, and Matthew A. Hammer. 2017. Hazelnut: a bidirectionally typed structure editor calculus. *ACM SIGPLAN Notices* 52, 1 (2017), 86–99. https://doi.org/10.1145/3009837.3009900

[5] Frank M. Shipman and Catherine C. Marshall. 1999. Formality considered harmful: Experiences, emerging themes, and directions on the use of formal representations in interactive systems. *Computer Supported Cooperative Work (CSCW)* 8 (1999), 333–352. https://doi.org/10.1023/A:1008716330212