The essential trick employed by Matthews and Findler is a syntactic one: they define their semantics in such a way that when it comes time to do the embedding, all they have to do is reinterpret the symbols in such a way that, magically, the correct new rules fall out! By example, here is a rule in the ML language:

$$\mathcal{E}[(+\ \overline{n_1}\ \overline{n_2})]_M \longmapsto \mathcal{E}[\overline{n_1+n_2}]_M$$

Notably, they use a different symbol here than the symbol defining evaluation contexts, which is simply a non-curly E. When they decide that the top-level evaluation context should be in Scheme, all they have to do is overload curly-E to be a blue E.

$$E[(+\ \overline{n_1}\ \overline{n_2})]_M \longmapsto E[\overline{n_1+n_2}]_M$$

One more thing to note: while the evaluation context is a blue E, the term expected to be written in the hole is RED! This is indicated by the little subscript M. So there is a difference between what the top-level evaluation context is, and what actually is being evaluated in the hole.

What's the idea? The idea is that we simply write our rules in a convenient style, and then overload the symbol so as to turn it into a rule on the FULL machine states we care about. I will describe this concretely below. First, the ML language.

$$E = [\,] \mid E\,e \mid \ldots$$

$$\mathcal{E}_\Sigma[!a]_T \longmapsto \mathcal{E}_\Sigma[\Sigma(a)]_T$$

Our goal is to give an expansion of curly E with subscript sigma. When curly E is inlined, we should be given a full rule for evaluating the entire threadpool. If we only ever evaluate the first thread, the rule looks like this:

$$\Sigma, (\Sigma, E[!a]_T) \cdots \longmapsto \Sigma, (\Sigma, E[\Sigma(a)]_T) \cdots$$

Thus:

$$\mathcal{E}_\Sigma[e]_T \triangleq \Sigma, (\Sigma, E[e]) \cdots$$

Where did the dots and the blue sigma come from? Well, as it turns out, the curly
E is parametrized over these elements as well: this was hidden when we wrote
the rules initially. We're indifferent to the choices here, "These rules work for any choice
of blue sigma and remaining threads."

There are a few remaining questions to be resolved:

- What is to be done about I-NoStepT? I-NoStepT is troublesome for deep reasons:
it's a negation, and can be cause for inconsistency if not properly stratified
(c.f. Datalog). Unfortunately, we cannot use the usual trick for side-stepping stuck
evaluation, because we desire our rules to be deterministic. I-NoStepT is well
defined in this case, however: it can be thought of as an infinite set of rules that
is added to our language to ensure progress. It seems to me that it would be more
standard for the threadpool rules to be nondeterministic, however!

- What do the IFC rules look like? Well, we could certainly define a curly E for blue
rules, but this time with blue sigma and the rest of the threads (the hidden parameters
in the red case). However, the fork rule must speak of the red sigma (violates abstraction!);
it is truly a rule that can *only* be specified after the embedding. I don't know if there's
a notational trick we can employ to work around this.