NEEL language proposal

Dan Dougherty

November 13, 2011 – 13 : 33

Our meeting on Friday showed me some things that might be confusing to people, so here I've changed the writing a little in hopes of making things clearer. The previous draft was mostly written "to myself" to make sure I had the ideas right, now that I know other people are going to read it I've added more details. (And fixed the typos!)

There should be no change in the syntax or semantics (unless I've introduced a bug!) so if this seems different from what you understood before, ask me.

- For ease of talking about things I've introduced the notion of a "**!**-expression". This is an expression of the form $\mathbf{!}q$ where q is a query. It is not itself a query (since we never evaluate it directly to get an answer) but it is useful to have a name for it.

- The *target variables* of a query are the variables that we match against primitive events in the input stream in order to compute the result of the query. A **!**-expression always has an empty set of target variables.

  We need the more general notion of *free variables* of a query or **!**-expression in order to start the main syntactic constraint (in the definition of the **seq** queries) that allows the semantics to be defined. The target variables of a query are each free variables, but when looking at a sub-query p of a larger query q there may be variables that are free in p which are used target variables only at a "higher level" in the syntax tree. In the same way, a variable occurrence may be free in p even though in the larger context q it is not free.

  Intuitively, the set of free variables in a query (or **!**-expression) p is the set of variables that you need to have bound to primitive expressions if you are in the middle of evaluating p.

  For example, in this artificial query

  $$\exists z \,.\, \mathbf{seq}((Ax), \mathbf{!}((By) : y.id = x.id), (Cz))$$

  the only target variable is x, and the only free variable is x. From the perspective of the subquery $((By) : y.id = x.id)$ the target variable is y, and both x and y are free. In the **!**-expression $\mathbf{!}((By) : y.id = x.id)$ x is free, but there are no target variables. In $\mathbf{seq}((Ax), \mathbf{!}((By) : y.id = x.id), (Cz))$ the target variables are x and z, and these are the free variables as well.

  This all seems complex and subtle but these ideas are the key details to manage in order to define (and implement!) the language.

1 Events and Histories

Fix a set of *event types* written $A, B \ldots$ With each event type is associated a set of *attributes*; each attribute has a corresponding domain of possible *values*. There are tw distinguished attributes, shared by all event types, called ts and te, taking values in (say) the natural numbers (or the reals?) modeling time.

Typically the domains will have predicates defined over them; for example we can compare timestamps by $\preceq$, etc. There may be other, domain-specific attributes.

A *complex event* is (simply) a set of events. If $S = \{e_1,\dots,e_n\}$ is a complex event, define the start and end times for S as follows (we abuse notation slightly by writing $S.ts$ and $S.te$): $S.ts = \min\{e_i.ts \mid 1 \le i \le n\}$ and $S.te = \max\{e_i.te \mid 1 \le i \le n\}$.

We can view the vocabulary of primitive events as a first-order logic signature viewing the event types and the attribute-domains as sorts and the attributes as functions. A *history* H is a structure for this language. Call the elements in the structure whose sorts are event types the *primitive events.* If e is a primitive event with attribute f defined then it is conventional write $e.f$ rather than $f(e)$. We impose the constraint that no two primitive events have the same timestamp. A history has a natural order on events induced by the time-stamps. [if we don't have $ts = te$ for primitive events, need to say more here...not now]

2 Syntax of Queries

For each event type A we have an infinite set of variables $v_1, v_2, \dots$. Let *Vars* denote the collection of all variables.

A *term* is built from variables and the attributes (viewed as function symbols). We consider constants to be 0-ary function symbols. Of course we impose the usual constraints that application of functions to terms must type-check.

An *atomic formula* is a predicate applied to an appropriate number of terms. Examples $x = y$, $x.ts \prec y.ts$, $x.name =$ "mo", etc.

Definition 2.1. Here we define the syntax of queries (and **!**-expressions). In fact we inductively define at the same time the *target variables* $TV(q)$ of a query or **!**-expression and the *free variables* $FV(q)$ of a query or **!**-expression.

- if q is a query, then $(\mathbf{!}q)$ is a **!**-expression.
  $TV(\mathbf{!}q) = \emptyset$ and $FV(\mathbf{!}q) = FV(q) \setminus TV(q)$
- a *simple query* is an expression of the form $(A\ x) : \alpha(x, \vec{y})$ where A is a primitive event type, x is a variable of type A and α is a boolean combination of atomic formulas.
  $FV((A\ x) : \alpha(x,\vec{y})) = \{x, \vec{y}\}$ and $TV((A\ x) : \alpha(x,\vec{y})) = \{x\}$.
- suppose that $q_1,\dots,q_n$ are queries; then
  - **and**$(q_1,\dots,q_n)$ is a query
  - $TV(\mathbf{and}(q_1,\dots,q_n)) = \bigcup\{TV(q_i) \mid 1 \le i \le n\}$
  - $FV(\mathbf{and}(q_1,\dots,q_n)) = \bigcup\{FV(q_i) \mid 1 \le i \le n\}$
- suppose that $q_1,\dots,q_n$ are queries; then
  - **or**$(q_1,\dots,q_n)$ is a query
  - $TV(\mathbf{or}(q_1,\dots,q_n)) = \bigcup\{TV(q_i) \mid 1 \le i \le n\}$
  - $FV(\mathbf{or}(q_1,\dots,q_n)) = \bigcup\{FV(q_i) \mid 1 \le i \le n\}$
- if q is a query and $x_1,\dots,x_n$ are variables then
  - $(\exists x \,.\, q)$ is a query

  - $TV(\exists x \,.\, q) = TV(q) \setminus \{x\}$
  - $FV(\exists x \,.\, q) = FV(q) \setminus \{x\}$

  Note: in the other draft I defined, for some reason, $(\exists x_1, \dots, x_n \,.\, q)$ all at once. But it is conceptually cleaner to view quantification "one variable at a time:" so you should treat $(\exists x_1, \dots, x_n \,.\, q)$ as syntactic sugar for n applications of this quantification syntax rule.

- suppose that for each $1 \leq i \leq n$, Q_i is either a query or a **!**-expression; then

  - $\mathbf{seq}(Q_1, \dots, Q_n)$ is a query, *provided* the following condition is met.

    To define the condition we need some notation. We say that the *positive part* of $\mathbf{seq}(Q_1, \dots, Q_n)$ is the subsequence of $(Q_1, \dots, Q_n)$ consisting of the queries among the Q_i, and the *negative part* of $\mathbf{seq}(Q_1, \dots, Q_n)$ is the subsequence of $(Q_1, \dots, Q_n)$ consisting of the queries r such that $(\mathbf{!}r)$ is one of the Q_i.

    Careful: what we are calling the "negative part" is a set of *queries,* not **!**-expressions. They are what you get *after* the **!**has been removed.

    Now our requirement is that

    for every query p in the positive part, and every r in the **!**part, $TV(r) \cap FV(p) = \emptyset$.

    That is, no query in the positive part can make reference to the target variable of a query in the negative part. (This is the key technical device to allow the semantics to be defined nicely! See Remark 3.3.

  - $TV(\mathbf{seq}(Q_1, \dots, Q_n)) = \bigcup\{TV(p_i) \mid 1 \leq i \leq k\}$
  - $FV(\mathbf{seq}(Q_1, \dots, Q_n)) = \bigcup\{FV(Q_i) \mid 1 \leq i \leq n\}$

Note that we always have $TV(q) \subseteq FV(q)$.

Definition 2.2. A query is *well-formed* if no variable is used twice as a target variable.

We need one more bit of notation. If q is a query $\mathbf{seq}(Q_1, \dots, Q_n)$ let us define $erase(q)$, the *erasure* of q, as the query obtained by removing the ! sign in front of each r that is in the negative part of q. Note that we do *not* perform this action recursively; we only remove the **!** at the "first level".

The next definition is a little tedious formally, but intuitively it is simple: $b(q)$ is the depth of nesting of **!**-operators in q.

Definition 2.3. If q is a query we define $b(q)$, the level of **!**-nesting of q, by

- $b(q) = 0$ if q is simple.
- $b(\mathbf{and}(q_1, \dots, q_n)) = \max\{b(q_i) \mid 1 \leq i \leq n\}$
- $b(\mathbf{or}(q_1, \dots, q_n)) = \max\{b(q_i) \mid 1 \leq i \leq n\}$
- $b(\exists x \,.\, q)) = b(q)$
- $b(\mathbf{seq}(Q_1, \dots, Q_n)) =$
  - if there are no **!** at the top level of any of the Q_i, in other words if $\mathbf{seq}(Q_1, \dots, Q_n) = erase(\mathbf{seq}(Q_1, \dots, Q_n))$, then $\max\{b(Q_i) \mid 1 \leq i \leq n\}$
  - otherwise $1 + b(erase(\mathbf{seq}(Q_1, \dots, Q_n))$

3 Semantics of Queries

3.1 Environments and Complex Events

If H is a history, an *environment over* H is a type-respecting partial function $\beta : Vars \to H$. The meaning of a query q over a history will be a set of environments.

Say that an environment β^+ is an *extension* of environment β if the domain of β^+ is a superset of that of β, and β^+ and β agree on their common environment. (This is the same as saying that β^+ is a superset of of β when functions are taken to be sets of ordered pairs...)

More precisely, let q be a query and let β be an environment over history H whose domain includes $FV(q)$. We will shortly define what it means for β to satisfy q in H, denoted $H, \beta \models q$. Anticipating this definition we define the meaning of a query as follows.

Definition 3.1 (Semantics of Queries)**.** Let q be a query and let H be a history. The meaning of q on H, written $q[H]$, is the set of those environments β_0 such that

- β_0 has domain $TV(q)$ and
- there is an extension β of β_0 such that $H, \beta \models q$.

The reason this is stated somewhat awkwardly, in terms of extensions, is explained in Remark 3.4 below.

It remains to define $H, \beta \models q$. Since this will be an inductive definition, we need to define $H, \beta \models q$ even when the domain of β includes free variables of q that are not target variables.

Definition 3.2. Let H be a history, let q be a query, and let β be an environment over H whose domain includes $FV(q)$. We define

$$H, \beta \models q$$

by induction on the level of nesting of negations, with a sub-induction on the size of q.

- Case: q is $(A\ x) : \alpha(x, \vec{y})$
  $H, \beta \models q$ if $\alpha(x, \vec{y})$ is true in H under β
- Case: q is $\mathbf{and}(q_1, \ldots, q_n)$
  $H, \beta \models q$ if for each i, $H, \beta \models q_i$
- Case: q is $\mathbf{or}(q_1, \ldots, q_n)$
  $H, \beta \models q$ if for some i, $H, \beta \models q_i$
- Case: q is $(\exists x \ . \ p)$
  Let β^{-x} be the environment obtained by removing x from the domain of β. Then $H, \beta \models q$ if there is an extension β^+ of β^{-x} such that $H, \beta^+ \models p$.
- Case: q is $\mathbf{seq}(Q_1, \ldots, Q_n)$
  - When $b(q) = 0$ q is of the form $\mathbf{seq}(q_1, \ldots, q_n)$.
    For each i let β_i be the restriction of β to the variables $TV(p_i)$. (Note that the domains of the β_i are pairwise disjoint by our assumption that no variable occurs more than once as a target variable.) Let C_i be the range of β_i; note that we may view each C_i as a complex event, so that $C_i.ts$ and $C_i.te$ are defined.
    Now we say that $H, \beta \models \mathbf{seq}(q_1, \ldots, q_n)$ if for each $1 \leq i < k$

1. $H, \beta \models q_i$ for each i [1] and
2. $C_i.te \prec C_{i+1}.ts$

– Next suppose $b(q) > 0$. Let the positive part of q be $\langle p_1, \dots, p_k \rangle$. Then we say that $H, \beta \models$ **seq**$(Q_1, \dots, Q_n)$ if

1. $H, \beta \models$ **seq**$(p_1, \dots, p_k)$, and
2. for no extension β^+ of β do we have $H, \beta^+ \models erase($**seq**$(Q_1, \dots, Q_n))$

Remark 3.3. We are in a position now to understand the restriction about target variables and free variables in the syntax of **seq** queries. We have to require in the defn that β be defined on all the free variables of the query, otherwise we couldn't refer truth of conditions back to the input history. But the target variables in the negative part are not free, so β need not be defined for the target variables of negated queries in q. This would seem to cause a problem for evaluating the positive part above. But all is OK since we said that these target variables from the **!**-part were not allowed to be free in the positive part. That's precisely why we needed that restriction.

Remark 3.4. Note the subtlety in the interactions between Definition 3.1 and Definition 3.2. In Definition 3.1 it is required that the domain of the answer-environment be precisely the set of target variables, while in Definition 3.2 we do not (cannot) make this assumption. This difference is required: to see why, consider the semantics of a query like **or**$((Ax), (By))$. Suppose H is $\langle a, b \rangle$. And consider the environment β that binds $x \mapsto a$ and $y \mapsto b$. Then certainly we want to say that $H, \beta \models$ **or**$((Ax), (By))$. But (it seems to me) we do *not* want β to be an answer to that query on H. Rather, the answers to the query on H should be the two environments: $x \mapsto a$ and $y \mapsto b$. This is what the wording of Definition 3.1 accomplishes.

A final note: It seems to be that there is a potential confusion for readers and users with our use of **and** and **or** as an operator on queries, and $\wedge$ and $\vee$ as operators on conditions inside of queries. I wonder if we should say something like **all** instead of **and**, and **some** instead of **or**?

[1] yes, we mean β, not β_i