recap

• 18/09
  • categories, monos, push-outs
  • expansive rewriting
  • forward propagation
  • homework assignment #1 [for Monday]

• 2/10
  • pull-backs, pull-back complements
  • restrictive rewriting
  • backward propagation
  • homework exercise #2

• 16/10
  • graph-based knowledge representation
  • the KAMI bio-curation system
  • choice of research papers to present
When we update $G$, we split the rule into two phases: **strict** & **canonical/propagation**.

- The **strict phase** updates $G$ in such a way that the result is still typed by $T$.
- The **propagation phase** completes the update of $G$ and afflicts the same update to $T$.

**Rule #1:** add an element that can be typed by $T$.

- Only need the strict phase of rewriting — no propagation to $T$.

**Rule #2:** add an element that cannot be typed by $T$.

- Only need the propagation phase — strict update is trivial.

**Rule #3:** merge two elements of different types in $T$.

- No strict phase but all the red and the elements turn purple! Side-effect.

**Combined rule:**

- We see the same side-effect but if we merge rules all of the same type no side-effect occurs and this can be performed in the strict phase of rewriting.
plan for my classes

• 18/09
  • categories, monos, push-outs
  • expansive rewriting
  • forward propagation
  • homework assignment #1
• 2/10
  • pull-backs, pull-back complements
  • restrictive rewriting
  • backward propagation
  • homework exercise #2
• 16/10
  • graph-based knowledge representation
  • the KAMI bio-curation system
  • choice of research papers to present
restrictive rewriting
intuitively

• to rewrite a graph $T$, need a rule $r : L \leftarrow R$, and a matching $m : L \rightarrow T$
  • intuition [from first week]: replace the image (via $m$) of $L$ in $G$ by $R$
  • $R$ may contain nodes/edges that are not in the image (via $r$) of $L$: these are deleted from $T$
  • $R$ may contain nodes that are the image (via $r$) of multiple nodes in $L$: these are cloned in $T$
• if $T$ types a graph $G$, i.e. we have $h : G \rightarrow T$, these changes may propagate
  • if nodes of $T$ are deleted, all their instances in $G$ must also be deleted [canonical]
  • we must specify which nodes of $G$, typed by a cloned node, are still uniquely typed [requires a choice]
• we are seeking an abstract means to express these aspects of rewriting
  • again uses the language of category theory…
duality
two for the price of one

• if \( C \) is a category, the opposite (or dual) category \( C^{\text{op}} \) is defined by:
  • the objects of \( C^{\text{op}} \) are the same as those of \( C \) and, if \( f : A \to B \) is an arrow in \( C \), \( f^{\text{op}} : B \to A \) is an arrow in \( C^{\text{op}} \)
  • the identity \( 1^{\text{op}} : A \to A \) in \( C^{\text{op}} \) is the identity \( 1 : A \to A \) in \( C \) and, if \( g : B \to C \) in \( C \), the composition in \( C^{\text{op}} \) is defined as \( f^{\text{op}} \circ^{\text{op}} g^{\text{op}} := (g \circ f)^{\text{op}} : C \to A \)

• if \( A \to A + B \leftarrow B \) is a co-product in \( C \), what is it in \( C^{\text{op}} \) ?

• all categorical concepts have a dual
  • the dual of initial object is terminal object: an object \( I \) with exactly one arrow from \( X \) to \( I \) [any \( X \)]
  • the dual of mono is epi: a pre-cancellable arrow
    • in Set, epis are precisely the surjective functions
  • the dual of a slice category is a co-slice category: arrows from \( T \)
  • the dual of co-product is product
  • the dual of push-out is…
pull-backs [not pull-ins!]
generalized products

• in the category of sets and inclusions, i.e. $\text{Set}^> / U$
  • if $B \subseteq D$ and $C \subseteq D$ then $A := B \cap C$ is the largest set such that $A \subseteq B$ and $A \subseteq C$
  • same as on the previous slide [where $D = U$]
• in the category of sets and functions, i.e. $\text{Set}$
  • if $f : B \to D$ and $g : C \to D$ then $A := \{(b,c) \in B \times C \mid f(b) = g(c)\}$ is the largest set with $g' : A \to B$ and $f' : A \to C$
    such that $g \cdot f' = f \cdot g$'
  • varying the set $D$ varies the result $A$
    • if $D = \{\cdot\}$, we get $A = B \times C$
    • as $D$ gets bigger, we pick out smaller sub-objects of $B \times C$ by violating $f(b) = g(c)$ in more cases

• the general notion of pull-back
  • given a pair of arrows $f : B \to D$ and $g : C \to D$
  • we want an object $A$ and a pair of arrows $g' : A \to B$ and $f' : A \to C$ such that $g \cdot f' = f \cdot g'$
  • satisfying: for any (other) object $X$ and arrows $g'' : X \to B$ and $f'' : X \to C$, there exists a unique arrow $h : X \to A$ such that $g'' = g' \cdot h$ and $f'' = f' \cdot h$

• what happens in $\text{Grph}$?
limits and co-limits

- pull-backs and push-outs are examples of (respectively) limits and co-limits
  - limit = “the largest … such that …” vs. co-limit = “the smallest … such that …”
  - formal definition requires new concepts: functor, natural transformation and functor category

- given categories $C$ and $D$, the functor category $D^C$ is defined by:
  - objects are functors $F : C \to D$, i.e. homomorphisms of categories
    - formally, two mappings — one from the objects and one from the arrows of $C$ to those of $D$ — satisfying, for all objects $A$ of $C$, $F(1_A) = 1_{F(A)}$ and, for all arrows $f : A \to B$ and $g : B \to C$ of $C$, $F(g \circ f) = F(g) \cdot F(f)$
  - arrows are natural transformations $\alpha : F \to G$, i.e. homomorphisms of functors $F,G : C \to D$
    - formally, a mapping from the objects $A$ of $C$ to arrows $\alpha_A : F(A) \to G(A)$ satisfying, for all arrows $f : A \to B$ of $C$, $G(f) \cdot \alpha_A = \alpha_B \cdot F(f)$

- limits, formally
  - a $J$-diagram is a functor $D : J \to C$ where $J$ is (typically) a finite category, a ‘shape’
    - the functor category $C^J$ is the category of $J$-diagrams
  - a cone to $D$ is an object $C$ of $C$ and arrows $\alpha_J : C \to D(J)$ such that $D(f) \cdot \alpha_J = \alpha_K$ [for $f : J \to K$ in $J$]
    - an arrow of $C^J$ of the form $\alpha : \Delta C \to D$ where $\Delta J : J \to C$ is the constant functor [all objects to $C$, all arrows to $1_C$]
  - a universal cone to $D$ is a cone $v : \Delta U \to D$ such that, for any cone to $D$, there is a unique arrow $f : C \to U$ of $C$ satisfying $v_J \cdot f = \alpha_J$ [for all objects $J$ of $J$]
  - a universal cone to $D$, if it exists, is called a limit of the diagram $D$ — and is unique up to unique isomorphism
  - co-limits use functors from $J^{\text{op}}$ to $C$

- provide convenient means for the definition of a large class of universal properties
  - however, not all universal properties can be captured by limits or co-limits…
pull-back complements

• in the category of sets and inclusions, i.e. $\textbf{Set}^> / \mathcal{U}$
  • if $A \subseteq B$ and $B \subseteq D$ then $C := D - (B - A)$ is the largest set such that $A \subseteq C$, $C \subseteq D$ and $A = B \cap C$

• general notion of pull-back complement
  • given a pair of composable arrows $g : B \to D$ and $f : A \to B$
  • we want an object $C$ and arrows $g' : A \to C$ and $f' : C \to D$ such that the square is a PB
  • satisfying...
  • NB: a PBC is neither a limit, nor a co-limit, although it has more the flavour of a limit

• in $\textbf{Set}$, PBCs do not always exist
  • a question of integer division
  • however, if $g : B \rightharpoonup D$, then $C := (D - \text{im}(g)) + A$ [can always integer divide by 1]
  • this is a sufficient [but not necessary] condition for PBCs to exist in $\textbf{Set}$: it has “PBCs over monos”

• in $\textbf{Grph}$, we also have all PBCs over monos
  • uses the definition in $\textbf{Set}$ for nodes and edges
restrictive rewriting
abstractly

• assume a category with all pull-back complements over monos
• a rule is an arrow \( r : L \leftarrow R \) and a matching is a mono \( m : L \twoheadrightarrow G \)
  • take the pull-back complement of \( r \) and \( m : R \twoheadrightarrow G \) and \( r' : G \leftarrow G' \)
  • \( G' \) is the updated version of \( G \)
  • \( m' \) is a matching of the RHS of \( r \) into \( G' \)
  • \( r' \) is the instantiation of \( r \) to \( G \)

• we have seen that \textbf{Set} and \textbf{Grph} have all pull-backs and all pull-back complements over monos
  • intuitive that \textbf{Set} / \( T \) and \textbf{Grph} / \( T \) also do
  • but how do we prove this categorically [i.e. for all slice categories]?
    • the proof for PBs is in the notes
    • optional [easy] exercise: prove this for PBCs
sesqui-push-out rewriting

- two rules with a common source object and a restrictive instance of the first
  - $L \leftarrow P \rightarrow R$ and $L \rightarrow G$
  - restrictive rewrite yields $P \rightarrow G$
  - expansive rewrite yields $R \rightarrow G^+$

- [less general] variants exist
  - single-push-out requires $L \leftarrow P$ : deletion [with side-effects] but no cloning
  - double-push-out requires $L \leftarrow P$ + concrete conditions to prevent deletion side-effects
- in practice, sesqui-PO provides all the typical rewriting operations required
  - includes deletion and merging side-effects — so rule application need not be reversible
  - this can be managed, without preventing cloning, e.g.
two technical points
pasting lemmas + mono preservation

• pasting for PBs and PBCs
  • if the left inner square is a PB then the outer rectangle is a PB if, and only if, the inner right square is a PB
  • if the left inner square is a PBC then the outer rectangle is a PBC if, and only if, the inner right square is a PBC

• mono preservation [homework #2]
  • in a PB square, if $f$ is a mono then so is $f'$

\[\begin{array}{cccc}
E & \longleftarrow & B & \longleftarrow & A \\
\downarrow & & \downarrow & & \downarrow \\
F & \longleftarrow & D & \longleftarrow & C \\
\end{array}\]

\[\begin{array}{cccc}
B & \longleftarrow & A \\
\downarrow & & \downarrow \\
D & \longleftarrow & C \\
\end{array}\]
another technical point

PBCs are stable under PB

• if we have a commutative cube where
  • the front face is a PBC
  • the left, bottom and back faces are PBs

then
  • the right and top faces are PBs
  • the back face is a PBC

• proof: homework #2
propagation

$G \rightarrow T$

• suppose we have a rule $r : L \leftarrow R$, a matching $m : L \rightarrow T$ and a typing $h : G \rightarrow T$
  • we want to apply $r$ to $T$ via $m$ — but…
  • we split $r$ into two phases:
    • the strict phase, which makes all changes to $T$ for which we can still type $G$
    • the propagation phase, which performs the remaining changes to $T$ and propagates these changes to $G$

• let us note that
  • by the pasting lemma, the two-stage rewrite of $T$ is equivalent to using $r$ directly
  • we can compute the propagation to $G$
    • by direct PB
    • by constructing the lifting of $r$ to $G$
  • these methods are equivalent by stability of PBCs under PB
• generalizes to arbitrary hierarchies
rule #1: clone an element and know how to retype G

\[ L \xrightarrow{\xi} \{\ldots\} \]
\[ T \xrightarrow{\{\ldots\}} \{\ldots\} \]

this homomorphism specifies how to retype G with T-

\[ G \xrightarrow{\{\ldots\}} L \xrightarrow{\{\ldots\}} \{\ldots\} \]
\[ T \leftarrow \{\ldots\} \]

only uses the strict phase of rewriting

rule #2: clone an element with propagation

\[ \{\ldots\} \]
\[ \{\ldots\} \]
\[ \{\ldots\} \]

only uses the propagation phase — the strict phase is trivial

Rule #3: delete an element

\[ L \xrightarrow{\{\ldots\}} \{\ldots\} \]
\[ T \leftarrow \{\ldots\} \]

in multi-set rewriting, no side-effect because the green elements of G have no incident edges — but, in general, this would delete an edge from (say) a green rule to a purple rule.
propagation

general case

• if we have multiple levels of typing, e.g. $G \rightarrow T \rightarrow U$
  • given a rule to be applied to $U$, we need one factorization for $T$ and another for $G$
  • these factorizations must be compatible: the strict phase for $G$ includes that for $T$

• if we have branching, e.g. $G \rightarrow U \leftarrow T$
  • we still need one factorization for $T$ and another for $G$
  • no compatibility is needed at this level — but will be required later if we “close the diamond”

• in general
  • all graphs above $U$ are susceptible to be modified by propagation
  • update source graphs first; then the pre-sources; &c.
    • this allows to perform in-place update
    • more efficient, at least for in-memory implementations
  • our implementation in Neo4j proceeds in the other direction
    • breaks typing at each rewrite
    • but we can exploit the QL to repair locally — and this performs propagation